PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 3 : G01B 11/00; G01N 21/47 G06G 9/00	A1	(11) International Publication Number: WO 84/ 01212 (43) International Publication Date: 29 March 1984 (29.03.84)

(21) International Application Number: PCT/US82/01277

(22) International Filing Date: 20 September 1982 (20.09.82)

(71) Applicant (*for all designated States except US):* CONTREX INC. [US/US]; 111 South Bedford Street, Burlington, MA 01803 (US).

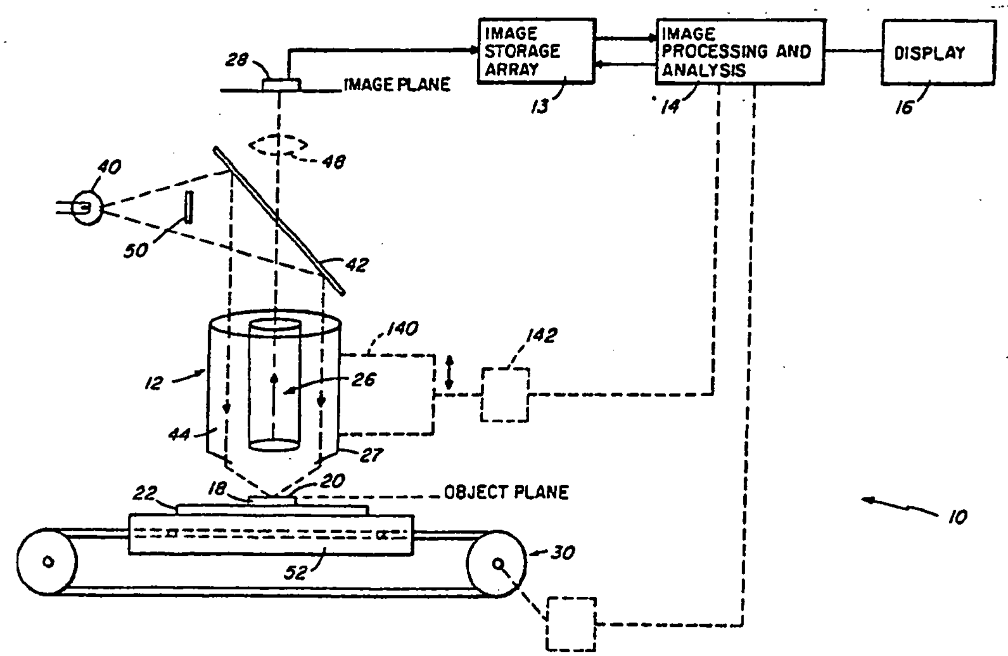
(72) Inventors; and
(75) Inventors/Applicants (*for US only*) : BRAUNER, Raul, A. [IL/US]; 33 Jodie Road, Framingham, MA 01701 (US). ESRIG, Paul [US/US]; 19 Spring Hill Road, Framingham, MA 01701 (US). LIFF, Harold [US/US]; 19 Douglas Road, Lexington, MA 02173 (US). ULLMAN, Shimon [IL/US]; 576 Washington Street, Apt. 2, Brookline, MA 02146 (US).

(74) Agents: WALPERT, Gary, A. et al.; Lahive & Cockfield, 60 State Street, Suite 510, Boston, MA 02109 (US).

(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), FR (European patent), GB (European patent), JP, LU (European patent), NL (European patent), SE (European patent), US.

Published
With international search report.

(54) Title: AUTOMATIC SEMICONDUCTOR SURFACE INSPECTION APPARATUS AND METHOD



(57) Abstract

An apparatus and method for the automatic inspection of a semiconductor wafer surface employs dark field illumination (12) for illuminating a wafer surface (20) to detect the material edges thereon. The surface (20) is scanned (30, 52) and an edge analysis (14) is performed for automatically determining material edge boundaries from the reflected energy spatial distribution. The edge boundaries are compared (126) with a reference pattern (120) and further analysis (128) determines the location of boundary disagreements between the material boundaries and the reference pattern. A report (130) is generated which can include for example, reticle cleaning or replacement information, defect locations, and defect classification including 'killer defects'.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	LI	Liechtenstein
AU	Australia	LK	Sri Lanka
BE	Belgium	LU	Luxembourg
BR	Brazil	MC	Monaco
CF	Central African Republic	MG	Madagascar
CG	Congo	MR	Mauritania
CH	Switzerland	MW	Malawi
CM	Cameroon	NL	Netherlands
DE	Germany, Federal Republic of	NO	Norway
DK	Denmark	RO	Romania
FI	Finland	SE	Sweden
FR	France	SN	Senegal
GA	Gabon	SU	Soviet Union
GB	United Kingdom	TD	Chad
HU	Hungary	TG	Togo
JP	Japan	US	United States of America
KP	Democratic People's Republic of Korea		

AUTOMATIC SEMICONDUCTOR SURFACE INSPECTION APPARATUS AND METHOD

Background of the Invention

The invention relates generally to the inspection of semiconductor wafers during manufacture, and in particular to a method and apparatus for the automatic inspection of semiconductor wafers during manufacture to determine the quality of the post-development photoresist and post-etch material structure.

Very Large Scale Integration (VLSI) technology and the immense packing density of the products resulting therefrom have, in recent years, required significant time to perform even small inspections of the semiconductor wafer as it is being processed. Such inspection typically requires a method for finding defects such as relatively small feature distortions and small size particulate contaminates.

The defects to be identified during semiconductor manufacture generally come from either the photolithographic process employed during manufacture or the properties of the photoresist with which the photolithographic process interacts. For example, the mask through which the semiconductor is exposed can have acquired a defect during handling or the photoresist can develop in a non-uniform manner thereby causing a defect to occur on the semiconductor surface. Other defects can occur due to particulate contaminates such as dirt particles which "land" on the semiconductor wafer surface during processing. Contaminates may also result from a "dirty" developer photoresist.

These d fects and contaminates, very small features in a relatively large inspection area, are important because they identify or help to correct potential problems prior to the completion of the manufacturing process. This early identification enables individual wafers containing critical defects to be disposed of at a stage prior to the completion of the manufacturing process. In addition, such information can be employed for monitoring the various stages of the fabrication process and can significantly affect the yield of the production line and hence the cost of manufacture. For example, early detection of a defect may allow the wafer to be reworked and the defect corrected.

Presently, inspection is performed either manually on selected semiconductor wafers or by machine. The manual or machine inspection processes often make decisions based only upon the relative feature differences between repeating patterns on the wafer surface.

An object of the invention is therefore an automatic inspection method and apparatus for semiconductor wafers which reliably and automatically identify sub-micron defects on the surface of the semiconductor element during manufacture. Other objects of the invention are a method and apparatus for the automatic inspection of semiconductor surfaces which detect both distortions or anomolies in the geometry on the surface as well as the presence of particulate contaminates. A further object of the invention is a method and apparatus for the automatic inspection of wafer surfaces which operate in real time and which

reduce the manufacturing cost of the fabrication process.

Summary of the Invention

The invention relates to a method and apparatus for the automatic inspection of a semiconductor wafer surface. The apparatus features an illumination system for illuminating the wafer surface to be inspected. Preferably, the illumination system employs dark field illumination for highlighting the material edges of the surface. A scanning system is provided for forming in a storage array a representation of the spatial distribution of illumination energy reflected from the surface. This spatial distribution represents, when dark field illumination is employed, the material edges of the wafer which has been illuminated. In a preferred embodiment of the invention the scanning system moves the wafer for scanning the inspection area while maintaining the optical illumination and receptor system stationary.

An edge analysis circuit automatically analyzes the reflected energy spatial distribution, which is represented in the array, for determining edge boundaries occurring on the wafer surface. A comparison circuit then compares the located edge boundaries (found by the analysis circuit) with a reference pattern which describes the expected geometrical layout of the wafer surface. The comparison circuit then determines the location of boundary disagreements between the analysis circuit edge boundaries and the reference pattern description. The boundary disagreements are then output, for example visually shown on a display, whereby the equipment user can personally view the defects.

In other aspects, the invention features a circuit employing an edge threshold level to discriminate between potential edge boundaries of different intensities, that is, to discriminate signal from noise. Thresholding acts as an amplitude filter. In addition, the boundaries are preferably spatially filtered (as described hereinafter) to form a more continuous edge pattern.

In the preferred embodiment, the apparatus further features a circuit for classifying the boundary disagreements and in particular for providing a class for "killer defects", that is, defects which prevent proper operation of a completed semiconductor circuit.

The apparatus further features circuitry for automatically determining, for a wafer having a repeating reticle pattern thereon (that is, a pattern formed using a reticle and which repeats on the wafer surface), whether a defect in the reticle has occurred and therefore whether the reticle should be cleaned or replaced. Furthermore, the apparatus provides circuitry for automatically repositioning the wafer surface for visual inspection of the surface at a selected boundary. In addition, circuitry is preferably provided to enable a more tolerant thereshold to be applied to matching edge corners on the wafer surface to the reference pattern.

In another aspect, the invention relates to a method for the automatic inspection of a semiconductor wafer surface. The method features the steps of illuminating the wafer surface to be inspected, preferably employing dark field illumination for

highlighting the edges of the surface. The method further featur s forming, in a storage array, a representation of the spatial distribution of illumination energy reflected from the surface; automatically analyzing the reflected energy spatial distribution for determining edge boundaries occurring on the wafer surface; comparing the edge boundaries found by the analysis step with a reference pattern description which describes the expected geometrical layout of the wafer surface; and then determining the location of boundary disagreements between the analysis edge boundaries and the reference pattern description. The boundary disagreements are then output, for example shown on a display, whereby the equipment user can view the defects.

In other aspects, the method features locating potential edge boundaries on the wafer using local differences in reflection values and then employing a threshold level to determine which edge boundaries are to be maintained and stored. The illustrated method also features spatially filtering the edge boundaries to form a more continuous edge pattern.

In the preferred embodiment of the invention, the method features classifying the various boundary disagreements, and in particular provides for a class for "killer defects", that is, defects which prevent proper operation of the semiconductor circuit. The method further features, in the illustrated embodiment, automatically determining, for a wafer surface having a repeating reticle pattern thereon, whether a defect in the reticle has occurred and therefore whether the reticle should be cleaned or replaced. Furthermore, the

illustrated method provides for automatic repositioning the wafer surface for visual inspection at a selected boundary and for providing a more tolerant threshold to be applied to matching corner edges of the wafer to the reference pattern.

Brief Description of the Drawings

Other objects, features, and advantages of the invention will appear from the following description taken together with the drawings in which:

Figure 1 is a schematic representation of the automatic inspection apparatus according to a preferred embodiment of the invention;

Figure 2 is a more detailed schematic of the image storage array according to a preferred embodiment of the invention;

Figure 3 is a diagrammatic representation of the convolution functions employed in connection with a preferred embodiment of the invention;

Figure 4 is a flow chart of the edge detection section according to a preferred embodiment of the invention;

Figure 5 is a flow chart of the edge pruning section according to a preferred embodiment of the invention;

Figure 6 is a flow chart of the edge comparison and report section according to a preferred embodiment of the invention;

Figure 7 is a diagrammatic representation of a lessening of tolerance with respect to a corner edge detection; and

Figure 8 is a schematic circuit diagram of the automatic detection apparatus according to a preferred embodiment of the invention.

Description of a Preferred Embodiment

General Structure

Referring to Figure 1, an automatic inspection apparatus 10 has an optical section 12, an image storage array 13, an image processing and analysis section 14, and a display section 16. In general operation, a semiconductor wafer 18 having a surface 20 to be inspected is mounted in a stable jig structure 22. The wafer surface is illuminated by the optical section 12. In particular, the preferred embodiment employs a 360° dark field presentation to highlight the edge structure present on the semiconductor surface. (In many applications bright field illumination can also be employed.) Reflected light is directed through the central image forming optics 26 of a microscope 27 (for example a Leitz Ergolus), and is focused on a photosensitive sensor 28 which may be for example a Fairchild Model CCD-133 having a 1024 x 1 linear element arrangement. The wafer surface is moved, by a step and repeat mechanism 30 attached to jig 22, in a direction transverse to the length of the optical array. Thereby, the image of the wafer surface scans across the sensor 28. The output of optical sensor 28 is stored in the storage array 13.

The image processing and analysis circuitry 14 accesses the stored data of array 13 and processes the data to locate edge boundaries on the semiconductor wafer surface. These edge boundaries may be photoresist edge conductors, or other material edges on the semiconductor surface. Circuitry 14 can be implemented in either hardware, software, or a combination of the two. When a software implemention is employed, illustrated circuitry 14 is implemented using a general

purpose digital computer, such as a Digital Equipment Corporation Model PDP-11/23 computer.

The image processing and analysis section 14 determines the locations of the edge boundaries on the wafer, smoothes and links those boundaries to form a more continuous edge pattern, and compares the edge boundary locations with a reference pattern of the design structure of the wafer surface. Any distortions from or disagreements with the expected pattern are flagged and become potential boundary disagreements. Each of the possible boundary disagreements is preferably classified and a disagreement list results therefrom. The group of boundary disagreements resulting from the analysis of a scanned frame by the image processing and analysis section is displayed, for example on a visual display. The visually presented information can describe the class and location of the defects or can automatically display the actual defects for visual inspection.

Background

As noted above, the present invention can be employed for monitoring a VLSI semiconductor fabrication. Typically, integrated circuits are fabricated by forming the circuit directly on a silicon crystal substrate. The substrate is typically a circular wafer, having a diameter of between three and six inches, and on each wafer will be fabricated several hundred complete circuits of the same type. Each complete circuit is on a die or chip which is generally rectangular in shape, several millimeters on a side. Following fabrication, the wafer is scribed to obtain the individual die for packaging or integration onto a more complex circuit.

Each of the die patterns is typically made by using either a mask or a reticle. The term "mask" is used to denote a patterned target which contains the patterns of all of the die on the wafer. The mask is generally a one-to-one image of the entire wafer and when a wafer is exposed through a mask, the entire wafer is effectively exposed at once. The term "reticle" is generally used to denote a patterned target which contains the pattern of at most a few die on the wafer. In the limit, the reticle may contain the pattern for only one die on the wafer. When using a reticle, the entire wafer is exposed by a step and repeat process. That is, one part of the wafer is exposed; the wafer is then stepped in a known direction; and the exposure is then repeated. By continuing the process, the entire wafer is covered with a repeating pattern. Thus, when exposing with a reticle, and especially a single die reticle, a defect in the reticle affects every group of die on the wafer made with that reticle. Since the VLSI technology is moving the industry away from masks and toward reticles, the inspection of the wafer surface for reticle defects is extremely important.

In the illustrated example, the semiconductor wafer 20 is assumed to have thereon a developed photoresist pattern. According to the invention, the photoresist pattern is being automatically inspected for defects such as geometric anomalies. Geometric anomalies occur, for example, as a result of errors or defects on the mask or reticle, particles which settle on or near the mask or reticle during exposure, particles which settle on the wafer during exposure, or development induced defects. In addition, the inspection process is designed to detect and locate

dimensional errors, which can occur when th photoresist pattern is geometrically correct but has certain critical dimensions which are out of specification, and particulate contaminates, that is, particles which fall onto a patterned photoresist.

In accordance with the invention therefore, the developed photoresist is illuminated, with dark field illumination in this preferred embodiment of the invention, for highlighting the edge information available on the wafer surface, and a digital image of the area to be inspected is acquired. The digital image is processed to generate or derive a description of the area being inspected in terms of the edges in the area. The edge information is presumed to completely define the boundaries of the photoresist and/or particulate contaminates lying thereon. In the case of the photoresist, the edges can be closely spaced to define conductors or can be spaced much further apart to define active areas such as the base or emitter of a transistor. With respect to particulate contaminates, the edges are spaced apart somewhat and form, generally speaking, a relatively ragged closed loop.

The Illumination System

The illustrated illumination system, as noted above, is a reflective or incident dark field illumination system which directs light energy from a source 40 onto a beam splitter 42 and through a mirrored collar 44 of the microscope 27 and onto the wafer surface 18 at an oblique angle. The incident illumination is then reflected back into and is collected by the microscope optics 26. The illuminated wafer surface image is focused by the microscope

objective and subsequent optics 48 (if needed) onto the surface of the optical sensor 28.

Because the wafer is opaque, it can only be imaged by light which is reflected from the surface. In dark field illumination, light is directed onto the object at a highly oblique angle through the mirrored collar 44 which surrounds the image forming lens system 26 of the objective. The light energy from source 40 is directed toward beam splitter 46 in an annulus configuration toward the sample semiconductor wafer. An opaque blocking member 50 is employed to prevent energy from being directed into the microscope image forming optics 26.

As is well known, the effect of using dark field illumination is to provide a highly specular reflection from an optically smooth, mirror-like surface such as is typical of the polished surface of an unpatterned silicon wafer. However with an optically rough surface, that is, one in which there are material discontinuities, the reflection is diffuse and in that case, reflective rays are scattered in all directions. Some of the reflected energy is captured by the microscope objective and the object appears bright at these areas. Thus, generally speaking, the unpatterned or optically smooth silicon substrate surface appears dark while the photoresist edges and particulate contaminates appear bright. (If bright field illumination had been employed, the silicon substrate would have appeared bright, while the photoresist edges would have appeared dark. Later image processing would proceed accordingly.)

Semiconductor Scanning

As noted above, the dark field image is formed by the microscope optics 26, and further focusing optics 48 if needed, at the image plane of the electro-optical sensor or detector 28. The sensor converts the reflected illumination incident thereon into an electrical signal which is later scaled and quantized into a discrete set of levels. Each level represents a small interval of illumination power and in the illustrated invention the total illumination range has two hundred and fifty-six levels. Sensor 28 is preferably a solid state sensor and in the illustrated embodiments is a linear photoresistive array. An alternate sensor could be a television-type vidicon camera.

The linear array approach thus employs a solid state image sensor having a plurality of distinguishable elements arranged in a rectilinear array. In the illustrated embodiment of the invention, the sensor 28 provides 1,024 distinguishable elements arranged in a straight line linear array. The area of the wafer imaged upon the array (a scan line) is thus spatially quantized into the 1,024 picture elements (pixels). Each pixel in the illustrated embodiment corresponds to 0.5 microns on the wafer surface. The illumination falls onto the array for a preset integration time during which light produced charge is collected in each of the distinguishable elements. At the end of the integration time, the charge accummulated at each element is read out and transduced into a voltage signal. The voltage is then scaled (or amplified) and quantized with the result being a spatial (1,024 elements) and voltage (256 levels) quantization of the line of the image.

In order to scan and produce an entire two-dimensional image, relative movement must be provided between the array and the wafer. Either the array must be moved across the stationary image or the image must be moved across the stationary array (a combination of the two could also be employed). As noted above, in the illustrated embodiment, the image is moved across the array. Thus, a mechanical stage 52 supporting the wafer 18 and jig 22 moves in a direction perpendicular to the array line under the control of a step and repeat mechanism.

An alternate approach, which reduces the movement required to produce a two-dimensional image of a selected surface area, is to employ an area array solid state sensor such as the Fairchild Model CCD-221. This sensor has a 488 x 380 element array.

No matter how the raw image is acquired, the resulting electrical data signals are stored in memory array 13 for later image processing. Referring to Fig. 2, in the illustrated embodiment of the invention, the storage array 13 has first and second random access memory (RAM) elements 54 and 56, one of which is being filled by the sensor 28 of scanning system while the other memory is being processed by the image processing and analysis section 14. Switches 58 and 60, which control the flow of data into and from elements 54 and 56, are preferably digital gating structures.

Image Processing

Referring now to the image processing and analysis section, the raw image data stored in memory array 13 represents the image as a two-dimensional

matrix of numbers. The "numbers" represent the image intensity across the spatial extent of the wafer surface being scanned. The image processing and analysis circuitry operates upon this raw image data to derive a description of the image in terms of potential edge boundaries (the edge finding procedure). Thereafter the edge boundary data (which identifies potential edge boundaries) is pruned or massaged to eliminate false boundaries and "clean-up" true boundaries (edge boundary pruning). Finally the edge boundaries are compared against a reference pattern, and defects or disagreement boundaries are recorded (edge boundary comparison).

<u>Edge Finding</u>

In the illustrated embodiment of the invention, the process of edge finding is implemented using convolution masks (or filters) operating along orthogonal axes. In the illustrated embodiment, these masks align with the horizontal and vertical axes with which most of the edges of the image will also align.

Referring to Fig. 3, ideally, a photoresist or other material edge, when illuminated with dark field illumination, produces a bell-shaped light intensity distribution (intensity as a function of distance) in a direction perpendicular to the edge. Thus, if the edge runs parallel to one orthogonal axis, the light distribution profile will be exclusively directed parallel to the other orthogonal axis.

An optically rough particle (such as a contaminant) will produce, in response to dark field illumination, a signal waveform (representing light intensity versus distance), having a rising and a

falling edge plus an intermediate region of relatively constant high intensity. To find and distinguish the material edges and the particle edges, two different convolution masks or patterns are employed. A peak finding convolution pattern w is designed to provide a zero crossing when the peak of an intensity distribution is crossed. A typical and preferred peak finding mask w has weighting factors w_i equal to -0.3, -0.1, 0, 0.1, 0.3 (for i = -2, -1, 0, 1, and 2, respectively), so that during the convolution process as defined by equation 1 below, a zero crossing indicates the center of the peak.

$$l_i = \sum_{k=-n}^{n} h_{i-k} w_k \tag{1}$$

where l_i represents a new sequence of numbers created by convolving an original sequence of numbers (h_i) with the weighting sequence w_i corresponding to the convolution mask. Thus, to find a horizontal edge, the convolution is performed upon a vertical line of data, and to find a vertical edge, the convolution is performed upon a horizontal line of data. For edges which are neither horizontal or vertical, a combination of the results of the horizontal and vertical peak finding convolution must be considered. It is important at this point to note, however, that the zero crossings resulting from the convolution process only provide the location of a potential edge point. Further processing (edge pruning) is required to determine whether the potential edge point is part of a material edge boundary.

The second convolution mask W, a step finding function, provides a data set for finding the edge

boundaries of a particle contaminant. A similar convolution approach is employed; however, the convolution mask is modified to provide a zero crossing where the edge has the appearance of a step with relatively wide pateaus extending from the step in both directions. Thus, the step finding convolution mask is designed to provide zero crossings at the center of an edge bounding a relatively large area or plateau. A typical step finding convolution mask, and the one employed in the illustrated embodiment, uses the weighting sequence W_i: -.4, -.1, 1, -.1, -.4 (for i = -2, -1, 0, 1, and 2, respectively). The results of using these convolution masks with a photoresist edge structure and a particle contaminate edge structure are illustrated in Figure 3.

Referring to Figure 4, which is a flow chart for that portion of the image processing and analysis section which relates to edge detection, the acquired image represented by block 60 is first spatially smoothed to help eliminate the noise "ripples" inherent in the digitization process of a noisy analog signal. The spatial filter provides low pass filtering which also helps eliminate invalid peaks due to system noise. The spatial filtering, represented by block 62, is applied along each of the orthogonal axes. A Gaussian function could be used, however it is much simpler to approximate the Gaussian by a weighting function having weights: 1/4, 1/2, 1/4. Thus, the value of each picture element intensity is replaced by an average equal to 1/4 of the previous value, plus 1/4 of the succeeding value, plus 1/2 of the present value. The smoothed data resulting from the operation indicated by block 62 is preferably stored in the same memory as the acquired raw image data.

Next, assuming that the orthogonal axes are the vertical and horizontal axes, the smoothed image is convolved in both the horizontal and vertical directions with the peak finding and step finding convolution functions respectively. This is indicated at blocks 64, 66, 68, and 70. The reuslt of the respective convolution processes is then searched for possible zero crossings. This is indicated at blocks 72, 74, 76, and 78. For each detected zero crossing, the strength of the crossing, is, for example, set equal to the peak amplitude of the other convolution function for that axis and within a small range of pixels of the zero crossing. The strengths are stored, as indicated at 80, 82, 84, and 86, preferably in the same storage array which originally stored the raw image data.

At this point, the strengths resulting from the step finding convolution are made positive. This is indicated at 88 and 90. Also, the zero crossings for the peak finding convolution result are reviewed by eliminating invalid zero crossings, i.e., those zero crossings representative of noise. These are generally weak zero crossings which do not have associated with them strong related zero crossings. This is represented by blocks 92 and 94 of Figure 4.

The edge detection process, by eliminating weak zero crossings of the peak finding convolution, discriminates between noise and potential photoresist edges. The strength measurement discriminator, in the illustrated embodiment, is a threshold value fixed prior to processing and in general depends upon the materials being employed. In other embodiments, the threshold value can be varied dynamically during processing to

take account of local variations in both noise and signal strength as a result of the semiconductor fabrication process.

The strength measurement for a zero crossing is, in the illustrated embodiment, the maximum value of the step finding convolution output within plus or minus one picture element of where the peak finding convolution output goes through zero. Importantly, the strength of the step finding convolution will not be "confused" with noise since it is not a peak finding element but instead effectively locates inflection points, that is, the position at which the first derivative of the image signal passes through a minimum or maximum.

The strengths are coded at 96, 98, 100, and 102 and are stored in coded fashion in the same memory used to first acquire the raw image data. Coding can be accomplished by allocating to each word of the array (one word representing one pixel), preassigned bits representing the vertical and horizontal axes, and the peak finding or step finding strength result. Alternately, the word can be divided to indicate whether the strength stored there is strong or weak, is the result of a step or peak finding convolution, and is for the horizontal or vertical axis.

During storage as indicated at block 104, the horizontal and vertical strengths for the peak finding convolutions, and the horizontal and vertical strengths for the step finding convolutions, are summed. This accommodates edge boundaries which are neither horizontal nor vertical but at an angle oblique thereto such as at a 45° angle.

The stored and coded zero crossing strengths are then analyzed to d tect valid edge boundaries and to discard invalid boundaries. This is referred to as the pruning process and is indicated at block 106 of Figure 4.

The Edge Boundary Pruning

Referring to Fig. 5, once the coded strength of the convolution edge detection process has been stored (block 108), and prior to forming the edge boundaries, the data must be further analyzed to remove invalid edge points. There is also a need to discriminate between edges representing, for example, a photoresist edge (Block 110) and those which are part of a particle contaminant edge (Block 112).

Thus, in the illustrated embodiment, if there is a peak finding convolution zero crossing within three picture elements of a step finding convolution zero crossing, then the step finding zero crossing is eliminated (Block 112). This occurs because it is assumed that the step finding zero crossing is erroneous, and it occurred in connection with and in the middle of a relatively wide photoresist area. Similarly, there might occur between two distant step finding convolution zero crossings, a peak finding convolution zero crossing. This can occur for example in the middle of a particle contaminant. In this case, the peak finding convolution zero crossing would be discarded (Block 110) although it is not generally necessary for later processing to do so. As a result of the pruning process therefore all that is left in the storage array 13, are edge boundaries because the discarded zero crossings will have been "zeroed".

The edge boundaries which remain however may or may not be complete and continuous. Thus, even though most of an dge boundary may be found, there can further be a gap in the edge boundary which should be filled in. The gap may occur because the edge point had a small strength. According to the preferred embodiment of the invention, these apparent discontinuities are smoothed and filtered by filling in the gaps between edge boundary points so that the edge is continuous along its boundary. This is indicated at block 116 of Figure 5.

Edge Boundary Comparison

Referring to Figure 6, the "pruned" edge boundaries are available to a comparison circuit as indicated at block 118. Initially, the "pruned" edge boundaries are aligned with a reference pattern (block 120). The reference pattern is provided from a reference data source such as a computer aided design (CAD) tape which is processed at 122 to provide data to the reference pattern, block 120. The alignment, indicated by block 124, is achieved primarily by "dead reckoning". That is, two relatively long edge boundaries, one parallel to one orthogonal axis and the other parallel to the other orthogonal axis, are selected in the reference pattern and the corresponding edge boundaries are "found" in the pruned edge boundary data memory. This process is practical only because the alignment of the wafer is known to within a few microns. Thus, the alignment search is carried out over a very small section of the memory and can be performed in a short time. The result of the alignment search is to provide horizontal and vertical offsets between the reference pattern and the stored data. Thereafter, as

indicated by block 126, edges in the reference block and the stored data are compared. Corresponding points, that is, points appearing in the same location in both patterns, are eliminated from the storage array 13, and, in the illustrated embodiment, non-corresponding points, that is, points in the reference pattern which do not appear in the storage array are written into the storage array at their appropriate locations. Points in the storage array which do not have a corresponding point in the reference pattern are kept. As a result, when the matching indicated by block 126 is completed, there results in the storage array 13 a set of disagreement boundaries which define distortions and particle contaminants, if any, on the image surface.

The disagreements are examined at block 128; and as a result, the disagreements or defects are classified. One particularly important class of defects or disagreements are those disagreements which materially affect proper operation of the semiconductor circuitry. These defects, if critical, are called "killer defects" and can be determined by defined areas of activity whose location can be provided by the reference pattern 120. Thus, a particle contaminant at a location spaced apart from the operating circuitry of the semiconductor wafer does not normally affect circuit operation whereas a contaminant on the circuit itself may cause the circuit to fail. In either case, a report is compiled, in the illustrated embodiment at block 130, and is provided to the display device 16 of Figure 1.

It is important to note, that a defect in one layer of a semiconductor structure can materially affect semiconductor circuit operation on another layer of the

structure. Therefore, the "defined areas of activity" provided by reference pattern 120 will relate not only to activity on the layer being formed, but also to the effect of a defect on a subsequently, or previously formed layer. In the illustrated embodiment of the invention, it is the CAD tape (or other reference source) which is processed at 122 to provide the multi-layer activity volumes in which a defect can have an adverse effect, and in particular where the defect is properly classified as a "killer defect".

In determining the "alignment" at block 124, it has been tacitly assumed that the "pruned" edges align with the horizontal and vertical axes as defined by the analysis process. This may not be the case however. Nevertheless, since the resolution of the system tends to be on the order of one-tenth of a micron per pixel, it has been found satisfactory to provide a plus or minus one picture element deviation in determining the alignment. A similar alignment tolerance has also been provided for determining whether other lines of the reference pattern and the stored detected edge boundaries "correspond" to one another.

A major concern which occurs during the comparison process of block 126 relates to the physical processes by which corners are formed during the semiconductor fabrication process. Due to the frequency response of the optical system employed in forming the photoresist corners, and further due to the effects of the chemical process by which the photoresist is layed down and developed, corners generally become rounded so that a truly "squared" edge does not occur. As a result, corners would almost always be "flagged" as a

defect absent any provision for loosening the tolerance of the system at the photoresist corner. As a result, referring to Figure 7, a loosening of th tolerance, or a window, is provided at the corner 132 defined by the reference pattern. The tolerance is illustrated by dashed lines 134, which allow the physical phenomena of a rounded corner represented by the dot-dash line 136 to be accommodated without being flagged a defect. Clearly other tolerance windows could be employed although the illustrated window is particularly easy to implement.

The illustrated embodiment can also be employed to implement automatic focusing of the optical system, by testing for the "sharpness" of the image at the optical sensor 28. The automatic focusing mechanism adjusts the microscope optics to provide as sharp an image as possible at the image plane of sensor 28. This can be accomplished for example by mounting the microscope illumination system on a jig as indicated by dotted lines 140 (Fig. 1) and moving the jig up or down under the control of a drive mechanism 142. The drive mechanism 142 is controlled by the image processing and analysis section 14.

As another feature, the step and repeat mechanism 30 can, under the control of the image processing and analysis section 14, reposition the semiconductor wafer to provide for a visual review of a defect on the semiconductor surface by the apparatus operator. The defect review can be accomplished using either the dark field illumination employed in connection with edge detection or bright field illumination for visual inspection.

As noted abov , it is the tr nd in today's VSLI technology to use a repeating pattern on a semiconductor wafer surface. Th apparatus herein is arranged to review the disagreements at block 128 for repeating patterns to find repeating defects, if any. Repeating defects are then reported as a possible and likely reticle defect which must be cured, for example, by cleaning the reticle or replacing it with a new element. This is accomplished at block 128 of Figure 6.

The entire analysis system can be implemented in either hardware or software. Preferably, hardware is employed since the throughput and process time can be decreased by use of special purpose hardware such as an array processor employing a pipeline processing approach. Nevertheless, a software implementation can also be satisfactory. The flow charts of Figures 4, 5, and 6 have been implemented in using a Digital Equipment Corporation PDP-11/23. The software programs, including interactive operating system programs, are attached hereto as Appendix A. While the programs themselves do not form part of the invention, they do provide one particular implementation of the concepts and structure of the invention. In addition, the invention can be implemented in hardware as described in detail hereinafter.

Hardware Implementation

As noted above, the automatic inspection system of the invention can also be implemented in hardware. Referring to Figure 8, the hardware embodiment employs a process control and sequence timing circuit 148 adapted to provide an orderly transition of the data from the microscope optics illustrated by block

150 to the eventual report generation and display. The process control and timing circuit can be a hardwired apparatus, as is well known in the art, adapted to fix the timing of a plurality of elements or can be a special or general purpose computer which provides greater flexibility in changing the timing and control of the apparatus.

The image from the illumination optics 150 is provided through the sensor element which forms part of an image acquisition section 152. The image acquisition provides the scanned image for storage in a dual memory storage array 154 corresponding to image storage array 13. The scanning of the wafer is under the control of a wafer scan control circuit 156 as is well known in the art which is interactive with the process control and sequence timing circuit 148.

The image, once stored, is continually modified within the storage element so that minimal additional RAM storage is needed. Therefore, the raw data stored in memory 154 is filtered using a spatial filtering network 158. The spatial filtering network is adapted to sequentially read out the raw data from memory 154, and to effectively low pass filter it as described above using its digitial hardwired circuitry.

After spatial filtering, the smoothed image data is convolved, by a convolution circuit 158 operating under the control of the control and timing circuit 148, for each of the convolution functions described in connection with Figure 3 so that a peak finding and step finding data is read into memory 154. The convolution circuit 158 is preferably built around

an array processor employing pipelined processing. The convolved (or filtered) data, in this illustrated embodiment, is then "pruned" for noise and similar anomalies by an edge pruning circuit element 162. The edge pruning circuit removes invalid edge points using the criteria described above in connection with Fig. 5. After the stored data in element 154 is "pruned", an edge boundary comparison circuit 164, also operating under the control of the process control and timing circuit 148, compares the data stored in the image storage array 154 with the reference model stored in a reference memory circuit 166. The output of the comparison, as described in connection with the flow chart of Figure 6, is stored back in storage array 154. As a result, there is found in storage array 154 the disagreement boundaries determined by a comparison of the processed scanned data with the reference model storage information. This stored information is then analyzed by the classification network 168. This network, after reference to memory 166, maps the boundary disagreements into classes depending in part upon the effect of the defect upon semiconductor operation, and provides detailed information regarding the defect and its classification to a report generating circuit 170. The report generating circuit provides a suitable format for either a visual or printed display. A display element 172 can thus be either a visual monitor which is preferred or a printer, or both. With reference to the generation of the reference pattern stored in memory 166, a CAD model is stored in memory, for example a disk memory 174 and the memory 174 is read and processed by a controller 176 for providing to the memory 166 both a suitable definition of the edge boundaries and a definition of the active volumes of the

final semiconductor structure which can be severely and adversely affected by defects in or near those reference boundaries.

The key to proper operation of the hardware is to provide sufficient timing and control via the process control and timing network 148 to enable the various elements to operate in a sequential manner and to use pipeline array processing as needed, such as, for example, the time consuming convolution process which involves a series of time consuming multiplications.

Additions, subtractions, deletions, and other modifications of this preferred embodiment of the invention will appear to those practiced in the art and are within the scope of the following claims.

```
: Install the VIDEO MONITOR task in low memory( up to 128kwords ).
dism dm3:
mcr mou dm3:
;set def 5,5
;run SY:[5,5]INIAP/par:ctask1
;set def 5,1
;run SY:[5,5]STARTAP/par:ctask1
;copy wafvis.txt tt0:
.ifins VIDEOT rem VIDEOT
.ifins MATCHT rem MATCHT
.ifins DEFECT rem DEFECT
.ifins CEDGET rem CEDGET
 ifins STAGET rem STAGET
ins/par:CTASK2/TA=VIDECT          SY:WFVIDEOT
ins/pri:65/TA=MATCHT              SY:WFMATCHT
ins/TA=DEFECT                     SY:WFDEFECT
ins/par:CTASK1/TA=CEDGET          SY:WFCEDGET
ins/TA=STAGET                     sy:WFSTAGET
run [5,1]wf2000c1/task=master
rem VIDEOT
rem MATCHT
rem DEFECT
rem CEDGET
rem STAGET
```

```
(*
        MASTER TASK: MASTERT.TSK
*)

ext MACOMM       ; Intertask communication support
ext PDPID        ; lsi-11 assembler symbol def
ext MAWIN        ; Memory management support
ext INSPLAN      ; The Inspection Plan and Inspection Status
ext IPSDBM       ; The Inspection Data Base Management
ext VDT          ; Miscellaneous terminal I/O routines
ext MAINIT       ; Master Initialisation

mvstr ( "master" , promstr )

$RESTART := base MAINIT

SAVE MASTERT




(* Global event flags for synchronization. TASK holds the names of the tasks
   with whom we are communicating in RAD50.  THESE MUST BE GLOBAL    *)
parameter TABLEN := 12.

record TASKTAB
  integer TAS ( 2 )
endrecord

integer SYNC1 ( TABLEN ) SYNC2 ( TABLEN ) TAPTR
TASKTAB TASK ( TABLEN )

with TASK ( 0 )
TAPTR OFF

(* Define executive directives to be used for tasking with Control *)

make 'WAIT rsxcall bytewd ( 2 , 41. )

make 'CLEAR rsxcall bytewd ( 2 , 31. )

make 'READ rsxcall bytewd ( 2 , 39. )

make 'SET rsxcall bytewd ( 2 , 33. )

make 'RCVD$ rsxcall bytewd ( 4 , 75. )

make 'SDRC$ rsxcall bytewd ( 7 , 141. )
```

```
make 'SDAT$ rsxcall bytewd ( 5 , 71. )

make 'WTLO$ rsxcall bytewd ( 3 , 43. )

make 'MRKT$ rsxcall bytewd ( 5 , 23. )

(* Execute a subroutine call. the arguments and subroutine name are in BUFF.
BUFF contains: TASK1 , TASK2 , 0 or -2 , #ARGS , arg1 , arg2 , .. argn , subrout
ine
    TASK1 and 2 make up the taskname of the caller  *)

define DOROUTINE
        integer BUFF ( 1 )
local
        integer ADR OFFST OFFST1
  OFFST := 4
  OFFST1 := 0
  if ( BUFF ( 2 ) )
    ptr ( BUFF ( 4 ) )
    OFFST1 := 1
    OFFST := OFFST + length ( ptr ( BUFF ( 4 ) ) ) / 2 + 1
  endif
  lookup ( ptr ( BUFF ( BUFF ( 3 ) + OFFST - OFFST1 ) ) ) ;; ADR := lastword
  DROP
  iter BUFF ( 3 ) - OFFST1
    ( BUFF ( I + OFFST ) )                    ; store args on stack




  loop
  exec ( ADR )
end

(* Wait for a logical or of event flags  *)

define WAITLO
local integer MASK
  MASK off
  iter TAPTR
    setbit ( SYNC1 ( i ) - 33 , ptr ( MASK ) )
  loop
 WTLO$ ( 2 , MASK )
end

define DELAY
        integer DTIM
   MRKT$ ( 23. , DTIM * 6 , 1 , 0 )
   WAIT ( 23. )
end

(* Return the index into the task table. called with the task name in RAD50  *)
define $TASGET integer
        integer T ( 1 )
  $TASGET on
  iter TABLEN
    with TASK ( i )
```

```
    if ( T ( 0 ) == TAS ( 0 ) and T ( 1 ) == TAS ( 1 ) )
        $TASGET := i
    endif
  loop
end

(* Return the index into the task table of a task. task name is in ascii  *)
define TASGET integer
        integer T ( 1 )
local   integer R ( 2 )
  TASGET on
  cluc ( T )
  if ( ascr5 ( T , R ) )
    TASGET := $TASGET ( R )
  endif
end

(* Primitive to send data to another task  *)
define SENDDATA
        integer INDEX , BUFF ( 1 )
  CLEAR ( SYNC2 ( INDEX ) )
  with TASK ( INDEX )
  SDAT$ ( TAS ( 0 ) , TAS ( 1 ) , BUFF , SYNC1 ( INDEX ) ) ;; :oerr
end

(* Wait for another task  *)
define WTASK




        integer TSKNAM ( 1 )
local
        integer INDEX
  detterm
  INDEX := TASGET ( TSKNAM )
  if ( INDEX == -1 ) print "Task not connected"
  else
    WAIT ( SYNC2 ( INDEX ) )
  endif
  atterm
end

(*  Connect to a task. It seems that RSX needs to have the terminal
    when the other task starts. we detach ourself and wait until the
    other task has initialised before we attach ourself again. Start
    up synchronization
    Call:  CONNECT ( "TSKNAM" SYNC1 SYNC2 )
    TSKNAM must be 6 characters. ie. CONNECT ( "DRW   " , 33 ,34 )  *)

define CONNECT
        integer TASKNAM ( 1 ) SYN1 SYN2
local
        integer BUFF ( 13 )
  cluc ( TASKNAM )
  if ( ascr5 ( TASKNAM , TASK ( TAPTR ) ) ==0 )
    print "Bad taskname"
  else
```

```
    detterm
    SYNC1 ( TAPTR ) := SYN1 ;; SYNC2 ( TAPTR ) := SYN2
    BUFF ( 0 ) := SYN1 ;; BUFF ( 1 ) := SYN2
    CLEAR ( SYNC1 ( TAPTR ) )
    CLEAR ( SYNC2 ( TAPTR ) )
    with TASK ( TAPTR )
    SDRCS ( TAS ( 0 ) , TAS ( 1 ) , BUFF , bytewd ( 16. 2 ) , 0 0 ) ;; ioerr
    WAIT ( SYNC2 ( TAPTR ) )
    increment TAPTR
    atterm
  endif
end

(* Send a buffer of data to the task we are connected to  BUFF ( 0 ) must
   be greater than 0. This can be used in the receiver as a code for what
   data has been sent. The buffer can be no longer than 13 words.  *)

define SEND
       integer T ( 1 ) BUFF ( 1 )
local  integer Z FLAGS ( 4 )
  Z := TASGET ( T )
  if ( Z == -1 ) print "Task not connected"
  else
    READ ( FLAGS )
    if ( getbit ( SYNC2 ( Z ) - 33. , ptr ( FLAGS ( 2 ) ) ) ) : If they are done
      SENDDATA ( Z , BUFF )
    else

      iter 3
        DELAY ( 100 )
        READ ( FLAGS )
        if ( getbit ( SYNC2 ( Z ) - 33. , PTR ( FLAGS ( 2 ) ) ) )
          SENDDATA ( Z , BUFF )
          exit
        else
          print str ( T ) , " is hung"
        endif
      loop
     endif
   endif
end

(* Call a subroutine that is in the task we are connected to.
   Call:    CALL TASK ROUTINE ARG1 , ARG2 , .. , ARGN
   where N <= 13 - ( #chars_in_routine / 2 + 1 ) - 1 - 1
                        string length          code #args
   this uses the code of 0 in the buffer       *)

define CALL command
       integer ARG
local
       integer BUFF ( 13 ) T ( 4 ) T1 ( 8 ) OFFST OFFST1
  detterm
  mvzer ( BUFF , 13 )
  BUFF ( 1 ) := cmdcnt - 3
```

```
    OFFST1 off
    OFFST := 2
    mvstr ( ARG , T )
    nxtarg
    mvstr ( ARG , T1 )
    nxtarg
    if ( ARG )
      BUFF ( 0 ) := -2
      BUFF ( 1 ) := cmdcnt - 2
      mvstr ( ARG , ptr ( BUFF ( 2 ) ) )
      OFFST1 := length ( ARG ) / 2 + 1
      OFFST := OFFST + OFFST1
    endif
    mvstr ( T1 , ptr ( BUFF ( cmdcnt + OFFST1 - 1 ) ) )
    nxtarg
    iter cmdcnt - 3
      BUFF ( I + OFFST ) := ARG
      nxtarg
    loop
    SEND ( T , BUFF )
    atterm
end

(* Call a subroutine in another task and wait for it to finish .
   Equivalent to CALL "TASK" "SUBROUTINE" 0 ;; WTASK ( "TASK" )  *)
define CALLW command
        integer ARG




local
        integer T ( 3 )
   mvstr ( ARG , T )
   exec ( base CALL )
   WTASK ( T )
end

(* Receive data from that task we are connected to and put it in a buffer
   Call:   RECEIVE ( BUFFER )  Note: if these routines are overlaid , BUFFER
   must be global.  The buffer must be at least 15 words. BUFFER contains:
      TASK1 , TASK2 , CODE , DATA     where TASK1 AND 2 make the name of the
   task which is sending the message. CODE is 0 if we are calling a routine,
   -1 if the other task is informing us of its rundown, and >0 if other data
   has been sent                                *)

define RECEIVE
        integer BUFF ( 1 )
local   integer Z
   WAITLO                                       ; wait for flag from any task
   RCVD$ ( 0 , 0 , BUFF ) ;; ioerr
   Z := $TASGET ( BUFF )
   CLEAR ( SYNC1 ( Z ) )
   if ( BUFF ( 2 ) == -1 )                      ; rundown
     SET ( SYNC2 ( Z ) )                        ; acknowledge receipt
     if ( BUFF ( 3 ) ==0 ) bye else return endif
   else
     if ( BUFF ( 2 ) ==0 or BUFF ( 2 ) == -2 )
```

```
            DOROUTINE ( BUFF )
        endif
        SET ( SYNC2 ( Z ) )
      endif
end

(* Inform the other task that we are stopping.
   Call:    RUNDOWN ( arg )
     arg = 0 if we want the other task to bye as well.
     arg ) 0 if we want the other task to stay alive or do something else
              before dying  *)

define RUNDOWN
        integer T ( 1 ) ARC
local
        integer BUFF ( 15 ) INDEX
   INDEX := TASGET ( T )
   if ( INDEX == -1 ) print "Task not connected"
   else
     BUFF ( 1 ) OFF
     if ( ARG ) BUFF ( 1 ) := ARG ;; ENDIF
     BUFF ( 0 ) ON
     SEND ( T BUFF )
   endif
end

(* Set up communications with the task that requested us.




    receives the two flags that the tasks will use for synchronization.
    and the name of the task with whom we are communicated.
    Any task that gets connected to must issue an INITREC command before
    proceeding.         *)

define INITREC
local
        integer BUFF ( 15 )
  detterm
  RCVDs ( 0 , 0 , BUFF ) ;; ioerr
  with TASK ( TAPTR )
  iter 2
    TAS ( I ) := BUFF ( i )
  loop
  SYNC1 ( TAPTR ) := BUFF ( 2 )
  SYNC2 ( TAPTR ) := BUFF ( 3 )
  SET ( SYNC2 ( TAPTR ) )
  increment TAPTR
end
```

```
(* The REGION Definition Block *)
record MEM_REC
        integer RGDB ( 0 )        ; Pointer to Region Definition Block.
        integer REGID             ; REGION ID
        integer REGSZ             ; REGION SIZE ( to be set )
        integer REGNM ( 2 )       ; REGION NAME IN RADIX50=<NO NAME>
        integer PARNM ( 2 )       ; NAME OF THE PARTITION IN RADIX50(to be set)
        integer REGST             ; STATUS: USE DEFAULTS(or to be set )
        integer REGPR             ; NOT PROTECTED AT ALL

(* The WINDOW Definition block *)
        integer WNDB ( 0 )        ; Pointer to Window Definition Block
        integer WNDAPR            ; HIGH BYTE HAS THE APR, LOW BYTE IS THE WINDOW
ID
        integer WNDADR            ; VIRTUAL BASE ADDRESS IN TASK'S VIRTUAL SPACE
        integer WNDSZ             ; WINDOW SIZE IN 32WORD BLOCKS
        integer WNDREG            ; REGION ID
        integer WNDOFF            ; OFSSET IN REGION IN 32 WORD BLOCKS
        integer WNDL              ; LENGTH TO MAP IN 32WORD BLOCKS
        integer WNDST             ; WINDOW STATUS WORD
        integer WNDSRB            ; SEN/RECEIVE BUFFER ADDRESS
endrecord

MEM_REC M_IPSDB          ;  Memory blocks for model access.

MEM_REC M_EDGE           ; for creation of EDGE IMAGE Region




MEM_REC M_MODEL          ; for creation of MODEL for MATCHING region

MEM_REC M_MATCH          ; for creation of IMAGE region for MATCHT and DEFECT.

(* Define the memory mngnment executives directives *)

make 'CRRG rsxcall bytewd ( 2 , 55. )
make 'DTRG rsxcall bytewd ( 2 , 59. )
make 'CRAW rsxcall bytewd ( 2 , 117. )
make 'MAPW rsxcall bytewd ( 2 , 121. )
make 'UMAPW rsxcall bytewd ( 2 , 123. )
make 'ELAW rsxcall bytewd ( 2 , 119. )


define CREGION
        integer REGNAM PARNAM REGSIZ
 REGSZ := REGSIZ
 ASCR5 ( REGNAM , REGNM ) drop
 ASCR5 ( PARNAM , PARNM ) drop
 REGST := 57K   ; attach it and allow all access
 CRRG ( RGDB )  ; create the region and attache it
 ioerr
end
```

```
define DREGION
 DTRG ( RGDB ) ,; ioerr
 if ( REGST )= 40000K ) print "Window unmaped" endif
end


define cwndow
       integer APR , WNRID , WNSIZ , WNOFF
 WNDAPR := urshift ( APR , 5 )   ; APR in the upper byte
 WNDSZ := WNSIZ          ; MUST BE LESS THAN 4K
 WNDREG := WNRID         ; THE REGION'S ID WHERE THE MAPPING TAKES PLACE
                         ; THIS IS KIND OF TRIKY NO. SO IT MUST BE FETCHED
                         ; FROM THE REGION DEFINITION BLOCK ( RGDB )
 WNDOFF := WNOFF         ; WINDOW OFFSET IN THE REGION
 WNDL off                ; TAKE THE DEFAULT. CAN BE CHECKED FOR THE ACTUAL
                         ; WINDOW SIZE AFTER THE CALL
 WNDST := 202K           ; MAP IT AND ALLOW WRITE ACCESS
 CRAW ( WNDB )  ; CREATE AND MAP THE WINDOW
 IOERR
END

(* INITRG ( REGNAM , PARNAM REGSIZ VADDR ) creates a named region and an
initial mapping of a 4k window at the begining of the region *)

define INITRG
       integer REGNAM PARNAME REGSIZ VADDR



 CREGION ( REGNAM , PARNAME , REGSIZ )  ; create a 32kwords dynamic region.
 CWNDOW ( VADDR , REGID , 200K , 0 )
                                ; create a window at VADDR  absolute address
                                ; of 4k words. Map it at the offset 0 in the
                                ; region
end
```

```
parameter MAX_FRAMES := 32
parameter MAX_SITES := 2
parameter MAX_PATTERNS := 1
parameter MAX_RETICLES := 2
parameter MAX_REVS := 1
parameter MAX_TEST_DIE := 2
parameter MAX_DIE_ROW := 10
parameter MAX_LAYERS := 1
parameter MAX_DEFECT := 14
parameter FALSE := 0
parameter TRUE := -1
parameter PRIMARY := 0
parameter CONFIRM := 1
parameter BRIGHT := 0
parameter DARK := 1

record   X_Y
  integer       X
  integer       Y
endrecord

record  ID
  integer       ROW
  integer       CLMN
endrecord

record  DEFECT




  integer       XCOM
  integer       YCOM
  integer       DELX
  integer       DELY
endrecord

record  DEFECT_BUFFER
  integer       #_DFCTS
  DEFECT        DEFECTS ( MAX_DEFECT )
endrecord

record  D_ROW
  integer       1ST_D_#
  integer       LAST_D_#
endrecord

record  F_DTL
  X_Y           F_SZ
  X_Y           F_OLAP
endrecord

record  F_TO_INSP
  integer       #_FS
  ID            FRAMES ( MAX_FRAMES )
  DEFECT_BUFFER F_DEFCTS ( MAX_FRAMES )
endrecord
```

```
record P_DTL
  char          P_DESCR ( 80 )
  integer       MIN_DEF_SZ
  integer       MIN_P_SZ
  integer       P_MAG
  integer       #_SITES
  X_Y           S_ORG ( MAX_SITES )
  X_Y           F_ORG
  F_DTL         F_DESCR
  F_TO_INSP     INSP_FR ( MAX_SITES )
endrecord

record  D_DTL
  X_Y           D_DIM
  integer       D_ST_HGT
  integer       D_AV_WDTH
  integer       #_PATTERNS
  P_DTL         D_PATTERNS ( MAX_PATTERNS )
 (* PATTERNS_TO_INSPECT  DIE_INSPECTION ( MAX_PATTERNS )  *)
endrecord

record  R_TO_INSP
  integer       #_TO_INSP
  ID            INSP_R ( MAX_RETICLES )
endrecord

record  R_DTL




  X_Y           R_DIM
  integer       R_ST_HGT
  integer       R_AV_WDTH
  D_DTL         RETICLE_DIE
(* DIE_TO_INSPECT  RETICLE_INSPECTION  *)
endrecord

record  L_DTL
  char          L_REV_# ( 80 )
  R_DTL         L_RETICLE
  R_TO_INSP     L_INSPECTION
endrecord

record  L_REVS
  char          L_DESCR ( 80 )
  integer       LAYER_#
  integer       #_REVS
  L_DTL         DTL_LAYER_REV ( MAX_REVS )
endrecord

record  PLAN_HDR
  char          PRODUCT_NAME ( 80 )
  integer       WAFER_SZ
  real          DIE_X
  real          DIE_Y
  X_Y           FLAT_TO_ORIGIN
  ID            REFERENCE_DIE
```

SUBSTITUTE SHEET

```
  integer        #_TEST_DIE
  ID             TEST_DIE ( MAX_TEST_DIE )
  integer        #_DIE_ROWS
  D_ROW          WAFER_MAP ( MAX_DIE_ROW )
endrecord

record  INSP_PLAN
  PLAN_HDR       HEADER
  integer        #_LAYERS
  L_REVS         LAYERS ( MAX_LAYERS )
endrecord


; DEFECT_BUFFER CONF_DEFECTS

; DEFECT_BUFFER REPT_DEFECTS


record  INSP_STATUS
        integer I-MODE          ; Primary or confirm
        integer STAGE_ERR       ; True , False
        integer STAGE_BUSY      ; True , False
        integer LENSE_BUSY      ; True , False
        integer FOCUS_BUSY      ; True , False
        integer ILLUM_BUSY      ; True , False
        integer REG_X




        integer REG_Y

        ID      MOD_RET ;
        integer MOD_SITE ; (1..15)
        integer MOD_FRAME ;
        integer MOD_ILLUM ; Bright , Dark
        integer MOD_MAGNF ; (1x .. 500x)
        integer MOD_LAYER
        integer MOD_PATTERN

        ID      CUR_RET ;
        integer CUR_SITE ; (1..15)
        integer CUR_FRAME ;
        integer CUR_ILLUM ; Bright , Dark
        integer CUR_MAGNF ; (1x .. 500x)
        integer CUR_LAYER
        integer CUR_PATTERN

        ID      DES_RET ;
        integer DES_SITE ; (1..15)
        integer DES_FRAME ;
        integer DES_ILLUM ; Bright , Dark
        integer DES_MAGNF ; (1x .. 500x)
        integer DES_LAYER
        integer DES_PATTERN

        X_Y     DES_DISPLAY     ; only screen coordinates
```

```
endrecord

record IPSDB_REC
  INSP_PLAN      INSP_PLN
  INSP_STATUS    INSP_DATA_BASE
endrecord
```

```
define STORE_IPSDB
        address NAME PLAN
local
        integer OUTCH
  OUTCH := open ( NAME , 'RWCT )
   wrs ( OUTCH , PLAN , SIZE IPSDB_REC ) DROP
  close ( OUTCH )
end

define READ_IPSDB
        address NAME PLAN
local
        integer INCH
  INCH := open ( NAME , 'R )
  rds ( INCH , PLAN , SIZE IPSDB_REC ) DROP
  close ( INCH )
end
```

```
(*      Miscellaneous terminal I/O routines for MASTERT *)

APUSH RADIX
OCTAL


(* GET AN INTEGER
 GETNUM ( PROMPT , DEFAULT )            *)
DEFINE GETNUM INTEGER
        INTEGER ARG1 ARG2
  GETNUM := ARG2
  PRINT STR ( ARG1 ) , "[" , #I 0 , ARG2 , "]: " , #Z
  IF ( RDLINE )
        IF ( ILITERAL ( LBUF ) )
                GETNUM := ILVAL
        ENDIF
  ENDIF
END

(*      Routines to set up a io/wait from the terminal   *)

  INTEGER  QIOW ( 0 )
        .WORD BYTEWD ( 12. , 3 )
        .WORD 1030K                     ; READ
        .WORD 0
        .WORD 24.                       ; EVENT FLAG
        .BLKW 3




        .WORD 1
        .BLKW 4

DEFINE TYI INTEGER
  QIOW ( 2 ) := CICH
  QIOW ( 6 ) := PTR ( TYI )
  TYI OFF
  RSXDIR ( QIOW ) ;; IOERR
END


(* GETFNUM ( REAL , PROMPT )    GET A REAL NUMBER    *)
DEFINE GETFNUM REAL
        REAL    FARG1
        INTEGER ARG3
  GETFNUM := FARG1
  PRINT STR ( ARG3 ) , "[" , #F 10 2 , FARG1 , "]: " , #Z
  IF ( RDLINE )
        IF ( ILITERAL ( LBUF ) )
                GETFNUM := FLOAT ( ILVAL )
        ENDIF
        IF ( RLITERAL ( LBUF ) )
                GETFNUM := RLVAL
        ENDIF
  ENDIF
END                                         *)
```

```
(* GETSTRING ( BUFFER , PROMPT )          *)
DEFINE GETSTRING
        INTEGER ARG1 ARG2
 WHILE ( not WORD )
   PRINT STR ( ARG2 ) , #Z
   RDLINE
 REPEAT
 MVSTR ( TBUF ARG1 )
END


(*  Get a string with a default
    GSTRING ( BUFFER , PROMPT , DEFAULT_STRING )        *)
define GSTRING
        integer BUFFER ( 1 ) PROMPT ( 1 ) DEFAULT ( 1 )
  mvstr ( DEFAULT , BUFFER )
  print str ( PROMPT ) , " [ " , str ( DEFAULT ) , " ] " , #z
  rdline
  if ( word )
    mvstr ( tbuf , BUFFER )
  endif
end

DEFINE YESNO INTEGER
        INTEGER ARG1
LOCAL INTEGER ANSWER




  PRINT STR ( ARG1 ) , " (Y/N): " , #Z
  ANSWER := TYI
  PRINT #A ANSWER
  YESNO := ( ANSWER and 137 ) == ASCII Y

END


INTEGER PAUSECHAR         ; Bucket for read in character

(*      PAUSE ROUTINES   *)

DEFINE VDT_IN
 address PROMPT
 PRINT str ( PROMPT ) , #Z
 PAUSECHAR := TYI
 PRINT #A PAUSECHAR
 CR
END


APOP
```

SUBSTITUTE SHEET

```
(* Initializion section of the master *)

(* Initialize region allocation *)
(* and Connect the module tasks *)

define MAINIT
 with M_EDGE
 CREGION ( "EDGIMG" , "EDGIMR" , 2000K )
 with M_MODEL
 CREGION ( "MODELR" , "GEN    " , 200K )
 with M_MATCH
 CREGION ( "MTCHIM" , "GEN    " , 2000K )
 with M_IPSDB
 INITRG ( "IPSDBR" , "GEN    " , 200K , 160000k )


 print "VIDEOT is being connected."
 CONNECT ( "VIDEOT" 41. 42. )   ; Initialize the Video Monitor Task VIDEOT
 print "MATCHT is being connected."
 CONNECT ( "MATCHT" 35. 36. )   ; Initialize the Registration and Mathcing
 print "DEFECT is being connected."
 CONNECT ( "DEFECT" 37. 38. )   ; Initialize the Defect Analysis Task
 print "CEDGET is being connected."
 CONNECT ( "CEDGET" 39. 40. )   ; Initialize the Edge Detection Task
 print "STAGET is being connected."
 CONNECT ( "STAGET" 33. 34. )   ; Initialize the SStage Positioning Task




;
end

define RUNDOWNALL
 RUNDOWN ( "VIDEOT" 0 )
 RUNDOWN ( "MATCHT" 0 )
 RUNDOWN ( "DEFECT" 0 )
 RUNDOWN ( "CEDGET" 0 )
 RUNDOWN ( "STAGET" 0 )
end

define BYE
 RUNDOWNALL
 bye
end
```

```
(* VIDOET Commands *)

define CGRABIM
        integer X0 Y0
 CALL "VIDEOT" "GRABIM" 0 X0 Y0
end

define CVDRAW
        integer FNAME X0 Y0
 CALL "VIDEOT" "VDRAW" FNAME X0 Y0
end

define CDRAW
        integer X0 Y0
 CALL "VIDEOT" "DRAW" 0 X0 Y0
end

define CMDRAW
        integer X0 Y0
 CALL "VIDEOT" "MDRAW" 0 X0 Y0
end

define CSAVE
        integer FNAME
  CALL "VIDEOT" "MSAVE" FNAME
end




define CVSAVE
        integer FNAME
  CALL "VIDEOT" "VSAVE" FNAME
end

define CFILLREG
        integer FEDGE
 CALL "VIDEOT" "FILLREG" FEDGE
end

define CREGFILL
        integer FEDGE
 CALL "VIDEOT" "REGFILL" FEDGE
end

define CWFMAP
        integer X0 Y0 SZ
 CALL "VIDEOT" "WFMAP" 0 X0 Y0 SZ ;  DISPLAY_WAFER_MAP
end

define CDISPMODEL
        integer X0 Y0
  CALL "VIDEOT" "DISPMODEL" 0 X0 Y0
end

define CBNDRCTS
        integer X0 Y0
```

```
  CALL "VIDEOT" "BNDRCTS" 0 X0 Y0
end

define STP-VIDEOT        ; disconnect "VIDEOT" task and attached to the terminal
        integer TERM
 CALL "VIDEOT" "STOPCO" TERM
 WTASK ( "VIDEOT" )
end


(* CEDGE Commands *)

define CEDGE
 CALL "CEDGET" "DOEDGE" 0
end

define CSTARTAP
 CALL "CEDGET" "START_AP" 0
 WTASK ( "CEDGET" )
end

define STP-EDGET         ; disconnect "EDGET" task and attached to the terminal
        integer TERM
 CALL "EDGET" "STOPCO" TERM
 WTASK ( "EDGET" )
end





(* MATCH Commands *)

define STP-MATCHT        ; disconnect "MATCHT" task and attached to the terminal
        integer TERM
 CALL "MATCHT" "STOPCO" TERM
 WTASK ( "MATCHT" )
end

define CREGISTER
 CALL "MATCHT" "REGISTER" 0
end

define CMATCH
 CALL "MATCHT" "MATCH" 0
end

define CGETIM
 CALL "MATCHT" "GETIM" 0
end

define COPYIM
 CALL "MATCHT" "COPYIM" 0
end

define CGETMODEL
        integer FNAME
```

```
 CALL "MATCHT" "GET_MODEL" FNAME
end


(* DEFECT Commands *)

define STP-DEFECT       ; disconnect "DEFECT" task and attached to the terminal
        integer TERM
 CALL "DEFECT" "STOPCO" TERM
 WTASK ( "DEFECT" )
end

define CDETECT
 CALL "DEFECT" "DETECT" 0
end


(* STAGE Commands *)

define STP-STAGET       ; disconnect "STAGET" task and attached to the terminal
        integer TERM
 CALL "STAGET" "STOPCO" TERM
 WTASK ( "STAGET" )
end

define CCALSTG  ; calibrate the stage




 CALL "STAGET" "CALSTG" 0
 WTASK ( "STAGET" )
end

define CSTGINI  ; calibrate the stage
 CALL "STAGET" "STGINI" 0
 WTASK ( "STAGET" )
end

define CSTAGEM  ; stage move according to inspection plan
 CALL "STAGET" "STAGEM" 0
end

define CILLSW   ; D/B field switch
 CALL "STAGET" "ILLSW" 0
end

define CFOCUS   ; autofocus
 CALL "STAGET" "FOCUS" 0
end

define CERRCOR  ; autofocus
 CALL "STAGET" "ERRCORR" 0
 WTASK ( "STAGET" )
end
```

```
ext MACOMANDS

                (* The demo section *)

integer DSKFLAG IMCNT   ; flag and image counter for EDGE image
                        ; saving on the disk
char    IMBASE ( 0 )
.text   "LNF"

integer RDSKFLAG RIMCNT ; flag and image counter for BRIGHT ( RAW ) image
                        ; saving on the disk
char    RIMBASE ( 0 )
.text   "RNF"

char    MDLBASE ( 0 )
.text   "LNF"

DSKFLAG on
RDSKFLAG on

integer         YESSTAGE ; &&&
YESSTAGE off

char ANSWER ( 20 )

define GET_LAYER




  DES_LAYER := GETNUM ( "Which mask level do you want to inspect " , 1 ) - 1 ; i
gnore user
; DISPLAY_INSPECTION_DIE
  CUR_LAYER := DES_LAYER
  with LAYERS ( CUR_LAYER )
  with DTL_LAYER_REV ( #_REVS - 1 )
  print "Mask revision number is .... " , str ( L_REV_# )
  print "Mask layer description is .. " , str ( L_DESCR )
  print "View screen to see preconfigured inspection die"
  print "       The blue reticle is the reference die"
  print "       The red reticles are the die:to inspect"
end

(* START_DEMO
Initializes the Inspection Data Base by reading from disk the IPSDB.DAT
and placing it in the common region IPSDBR. The Inspection Status is
Initialized.
*)
define START_DEMO
  IMCNT off
  with M_IPSDB
  GSTRING ( ANSWER , "Please input Product Name to be inspected" , "SEMIEAST" )
  READ_IPSDB ( "IPSDB.DAT" , WNDADR )
  ptr ( IPSDB_REC ) := WNDADR
  with INSP_DATA_BASE
  with INSP_PLN
  with HEADER
```

```
   print "Wafer size is ......... " , WAFER_SZ , " mm"
   print "Die width is .......... " , DIE_X , " microns"
   print "Die height is ......... " , DIE_Y , " microns"
   GET_LAYER
   CWFMAP ( 96. 383. 128. )
   mvstr ( 'LNF , IMBASE )
   mvstr ( 'RNF , RIMBASE )
   end

define SHOW_ME
   iter #_SITES
     with INSP_FR ( i )
     iter #_FS
       with F_DEFCTS ( i )
       if ( #_DFCTS )
         DES_SITE := j
         DES_FRAME := i
         with DES_RET
           ROW := INSP_R ( 0 ) : ROW
           CLMN := INSP_R ( 0 ) : CLMN
         CSTAGEM
         CFOCUS
         VDT_IN ( "Hit (return) to continue " )
           ROW := INSP_R ( 1 ) : ROW
           CLMN := INSP_R ( 1 ) : CLMN
         CSTAGEM




         CFOCUS
         print "Hit (return) to continue"
         VDT_IN ( "Hit ! to abort inspection:  " )
         if ( PAUSECHAR == ascii ! ) exit     endif
       endif
     loop
     if ( PAUSECHAR == ascii ! ) exit endif
   loop
end

define DOPRINTS
  och := open ( 'REPORT.DAT , 'wn )
  print #t 30 , "C O N T R E X "
  print #t 28 , "Wafervision  2000"
  print
  print
  print #T 30 , "Defect Report"
  iter 3
   print
  loop
  print "Product Name: " , str ( PRODUCT_NAME )
  print "Layer Description: " , str ( L_DESCR )
  print "Layer Revision Number: " , str ( L_REV_# )
  iter 3
   print
  loop
```

```
    print #T 25 , "Repeating Defects"
    print
    print #T 10 , "Number" , #T 30 , "X Location" , #T 50 , "Y Location"
    print #t 30 , "( Microns )" , #t 50 , "( Microns )"
  end

  define REPORT_DEMO
  local
          integer DFCTCNT TEMP TEMP1
          real CONVFACT
    DOPRINTS
    DFCTCNT := 1
    CONVFACT := .5 * 80.0 / FLOAT ( CUR_MAGNF )    ; .5 microns/pix @ 80X , adjust
    with F_DESCR
    iter #_SITES
      with S_ORG ( i )
      with INSP_FR ( i )
      iter #_FS
        with FRAMES ( i )
        with F_DEFCTS ( i )
        iter #_DFCTS
         with DEFECTS ( i )
          TEMP := fix ( float ( XCOM ) * CONVFACT ) + ( F_SZ : X * ROW ) + X
          TEMP1 := fix ( float ( YCOM ) * CONVFACT ) + ( F_SZ : Y * CLMN ) + Y
          print #T 5 , DFCTCNT , #T 29 , TEMP , #T 49 , TEMP1
          increment DFCTCNT
        loop




      loop
    loop
    close ( och )
    och on
  end


  define GETEDGES
  local
          char PNAME ( 30. )
          CALL "VIDEOT" "ACOMSG" 0
          WTASK ( "VIDEOT" )
    if ( DSKFLAG )
          print str ( IMBASE ) , #p 60k , #i 2 , CUR_FRAME , #n
          encode ( PNAME )
          CFILLREG ( PNAME )
          WTASK ( "VIDEOT" )
    endif
  end


  define GETBFIMG
          integer X0 Y0
  local
          char PNAME ( 30. )
    if ( RDSKFLAG )
          print str ( RIMBASE ) , #p 60k , #i 2 , CUR_FRAME , #n
```

```
        encode ( PNAME )
        CVDRAW ( PNAME  , X0 Y0 )
        WTASK ( "VIDEOT" )
  endif
end

define 1STFOCUS
if ( not RDSKFLAG )
        DES_ILLUM := DARK        ; MUST START WITH DARK ILLUMINATION!!!!!
        CILLSW             ; switch illumination
endif

if ( YESSTAGE ) ; &&&
        CSTAGEM            ; move the stage to the very first position
; &&&   CFOCUS             ; and focus on it
else
        CUR_FRAME := DES_FRAME
endif
end

define DBF       ; display bright field
        CALL "VIDEOT" "GBAR" 0 32 256. 0 256 0
        WTASK ( "VIDEOT" )
if ( not RDSKFLAG )
        DES_ILLUM := BRIGHT      ; meanwhile change illumination to bright
        CILLSW ;; WTASK ( "STAGET" )
endif




if ( RDSKFLAG )
        GETBFIMG ( 32 0 )
else
        CGRABIM ( 32. 0 ) ;; WTASK ( "VIDEOT" )
endif
        DES_ILLUM := DARK
end

define RTMOD
; preserve the current status in MOD_ status for inspection
        MOD_LAYER := CUR_LAYER
        MOD_PATTERN := CUR_PATTERN
        MOD_RET := CUR_RET
        MOD_SITE := CUR_SITE
        MOD_FRAME := CUR_FRAME
end

define GMLRGR    ; get and display the model and register
local
        char PNAME ( 30. )
          print str ( MDLBASE ) , #p 60k , #i 2 , MOD_FRAME , ".MDL" , #n
          encode ( PNAME )
          CALL "VIDEOT" "IPRMSG" 0 ;; WTASK ( "VIDEOT" )
          CGETMODEL ( PNAME ) ;; WTASK ( "MATCHT" )
          CDISPMODEL ( 352. 0 ) ;; WTASK ( "VIDEOT" )
; back to EDGE detection"
```

```
          VTASK ( "CEDGET" )     ; Before registration, EDGE must finish
          COPYIM VTASK ( 'MATCHT )       ; Copy EDGIMG to MTCHIM.
          CREGISTER
          VTASK ( "MATCHT" )
end

define MCHDEF   ; matchin and defect analysis
; COMPLETE DEFECTS
          CALL "VIDEOT" "MDLMSG" 0
          VTASK ( "VIDEOT" )
          CMATCH
          VTASK ( "MATCHT" )
          CALL "VIDEOT" "DFTMSG" 0
          VTASK ( "VIDEOT" )
          CDETECT ;; VTASK ( "DEFECT" )
          CBNDRCTS ( 32. 0 ) ;; VTASK ( "VIDEOT" )      ; Display the defects
          VDT_IN ( "Please contemplate and evaluate!!!" )
end

define CONFIRM_INSPECT
  iter #_SITES
    with INSP_FR ( i )
    iter #_FS
      with F_DEFCTS ( i )
      if ( #_DFCTS )

        DES_SITE := J





        DES_FRAME := I
;       if ( i + j ==0 )
        1STFOCUS
;       else
        VTASK ( "STAGET" )
        CALL "VIDEOT" "LMAG" 0
        VTASK ( "VIDEOT" )
if ( DSKFLAG )
        GETEDGES
else
        CGETIM          ; get the DF image
        VTASK ( "MATCHT" )
        CSTARTAP        ; start the array processor
        CEDGE           ; EDGE DETECTION
        VTASK ( "CEDGET" ) ; print "; while EDGE is working do the following"
endif
        DBF      ; display BF image
        RTMOD    ; set the model infor for real-time work
        GMLRGR   ; get and display the model and register
        MCHDEF   ; match and defect analysis
;       endif
        endif
    loop
    if ( PAUSECHAR == ascii ! ) exit endif
  loop
end
```

```
define PRIMARY_INSPECT
  iter #_SITES
    DES_SITE := i
    with INSP_FR ( DES_SITE )
    print "frames to inspect = " , #_FS
    iter #_FS
      DES_FRAME := I
;        if ( i + j ==0 )
          1STFOCUS
;        else
          WTASK ( "STAGET" )
      CALL "VIDEOT" "LMAG" 0
      WTASK ( "VIDEOT" )
if ( DSKFLAG )
        GETEDGES
else
        CGETIM          ; get the DF image
        WTASK ( "MATCHT" )
        CSTARTAP        ; start the array processor
        CEDGE           ; EDGE DETECTION
        WTASK ( "CEDGET" ) ; print "; while EDGE is working do the following"
endif
        DBF      ; display BF image
        RTMOD    ; set the model infor for real-time work
        GMLRGR   ; get and display the model and register
        MCHDEF   ; match and defect analysis
;       endif




    loop
  loop
end

integer NOFRAMES        ; &&& FOR TESTING PURPOSE ONLY
integer NOSITES         ; &&&
NOFRAMES := 6 ; &&& FOR TESTING PURPOSES ONLY
NOSITES := 1

define INSPECT_DEMO
  with L_INSPECTION
  with L_RETICLE
   with RETICLE_DIE
    DES_PATTERN := 0    ; initialise the pattern
    with D_PATTERNS ( DES_PATTERN )
    #_SITES := NOSITES ; &&&
    iter #_SITES
      with INSP_FR ( i )
      #_FS := NOFRAMES ; &&& FOR TESTING PURPOSES ONLY
      iter #_FS
        with F_DEFCTS ( i )
        #_DFCTS := 0                        ; initialize number of defects
      loop
    loop
 with DES_RET
   ROW := INSP_R ( 0 ) : ROW
   CLMN := INSP_R ( 0 ) : CLMN
```

```
    I-MODE := PRIMARY
    CALL "VIDEOT" "SHOWDIE" 0
    WTASK ( "VIDEOT" )
    PRIMARY_INSPECT
    if ( YESNO ( "Do you want to confirm the frames processed so far?" ) )
      ROW := INSP_R ( 1 ) : ROW
      CLMN := INSP_R ( 1 ) : CLMN
      I-MODE := CONFIRM
      mvstr ( 'CLN , IMBASE )
      mvstr ( 'RCL , RIMBASE )
      CALL "VIDEOT" "GBAR" 0 0 288. 256. 90. 0
      WTASK ( "VIDEOT" )
      CALL "VIDEOT" "SHOWDIE" 0
      WTASK ( "VIDEOT" )
      CONFIRM_INSPECT
    endif
end

define DEMO
    CALL "MATCHT" "WINDOW" 0 4 4
    START_DEMO
    CUR_ILLUM := BRIGHT
    if ( YESNO ( "Do you want to calibrate the stage?" ) )
        CCALSTG
    endif
    CALL "VIDEOT" "VDRAW" "JOE" 352. 256.
    WTASK ( "VIDEOT" )




    INSPECT_DEMO
    REPORT_DEMO
;    CALL "VIDEOT" "ENDMSG" 0
    print "End of Demo
end

DEFINE HAROLD_DEMO
  START_DEMO
  CGETMODEL ( "LNF03.MDL" )
  WTASK ( "MATCHT" )
  CALL "VIDEOT" "HAROLD_DEMO" 0
  WTASK ( "VIDEOT" )
END
endfile
```

```
(* *************************************************************************
        VIDEOT.MG - THIS MODULE LOADS ALL OF THE MODULES USED IN "VIDEOT.TSK"
 ************************************************************************* *)

ext     MAKLEX
ext     PDPID
ext     DMISC
ext     VIDREG
ext     FLUT
ext     FXMON
ext     22BADDR
ext     VIDDISP
ext     FDMACO
ext     VIDCOM
ext     [5,1]INSPLAN
mvstr ( "videot" , promstr )

parameter  GRNGL := 1
parameter  REDGL := 2
parameter  BLUGL := 3

define LLUSETUP
  TPLANE := 3
  GPLANE := 3




  IPLANE := 252.
  DSCHAN ( TPLANE , GPLANE , IPLANE )
  CLRMAP ( 0 )
  iter 256.
    DSLLU ( GREEN + GRNGL + I , 255 , GREEN + GRNGL + I , 255 )
    DSLLU ( RED + REDGL + I , 255 , RED + REDGL + I , 255 )
;    DSLLU ( GREEN + BLUGL + I , 128. , GREEN + BLUGL + I , 128. )
;    DSLLU ( RED + BLUGL + I , 255 , RED + BLUGL + I , 255 )
    DSLLU ( BLUE + BLUGL + I , 255 , BLUE + BLUGL + I , 255 )
    DSLLU ( GREEN + I , I , GREEN + I , I )
    DSLLU ( RED + I , I , RED + I , I )
    DSLLU ( BLUE + I , I , BLUE + I , I )
  loop ( 4. )
end

define INITD
  DSOPN ( 8 ) drop
  DSPLD
  DSCLR ( 255. ) DSCXY ( 0 0 )
  LLUSETUP
end

save    VIDEO
```

```
(* THE MAKE OF THE LEXIDATA 3400 LIBRARY ROUTINES *)
APUSH RADIX
octal
        MAKE    'DSOPN  RSXFUNC 60
        MAKE    'DSCLS  RSXFUNC 62
        MAKE    'DSCFG  RSXFUNC 64
        MAKE    'DSMRG  RSXFUNC 66
        MAKE    'DSZOM  RSXFUNC 70
        MAKE    'DSMOV  RSXFUNC 72
        MAKE    'DSLLU  RSXFUNC 74
        MAKE    'DSLWT  RSXFUNC 76
        MAKE    'DSLRD  RSXFUNC 100
        MAKE    'DSCHAN RSXFUNC 102
        MAKE    'DSVEC  RSXFUNC 104
        MAKE    'DSCLR  RSXFUNC 106
        MAKE    'DSCIR  RSXFUNC 110
        MAKE    'DSPNT  RSXFUNC 112
        MAKE    'DSLIM  RSXFUNC 114
        MAKE    'DSPUT  RSXFUNC 116
        MAKE    'DSGET  RSXFUNC 120
        MAKE    'DSOWT  RSXFUNC 122
        MAKE    'DSIWT  RSXFUNC 124
        MAKE    'DSRNW  RSXFUNC 126
        MAKE    'DSRNR  RSXFUNC 130
        MAKE    'DSSAO  RSXFUNC 132
        MAKE    'DSTXT  RSXFUNC 134
        MAKE    'DSCSL  RSXFUNC 136




        MAKE    'DSCER  RSXFUNC 140
        MAKE    'DSCLD  RSXFUNC 142
        MAKE    'DSCXY  RSXFUNC 144
        MAKE    'DSBLIN RSXFUNC 146
        MAKE    'DSBLOC RSXFUNC 150
        MAKE    'DSBLR  RSXFUNC 152
        MAKE    'DSGXY  RSXFUNC 154
        MAKE    'DSPLD  RSXFUNC 156
APOP
```

```
(*      Miscellaneous routines for DEMO  *)

APUSH RADIX
OCTAL


DEFINE LIMIT INTEGER
        INTEGER ARG1 ARG2 ARG3
  LIMIT := MAX ( ARG2 , MIN ( ARG1 , ARG3 ) )
END

DEFINE GETNUM INTEGER
        INTEGER ARG1 ARG2
  GETNUM := ARG2
  PRINT STR ( ARG1 ) , "[" , #I 0 , ARG2 , "]: " , #Z
  IF ( RDLINE )
        IF ( ILITERAL ( LBUF ) )
                GETNUM := ILVAL
        ENDIF
  ENDIF
END

DEFINE BEEP
  PRINT #A 7 , #Z
END

(*     Routines to set up a io/wait from the terminal   *)




  INTEGER  QIOW ( 0 )
        .WORD BYTEWD ( 12. , 3 )
        .WORD 1030K                     ; READ
        .WORD 0
        .WORD 24.                       ; EVENT FLAG
        .BLKW 3
        .WORD 1
        .BLKW 4

DEFINE TYI INTEGER
  QIOW ( 2 ) := CICH
  QIOW ( 6 ) := PTR ( TYI )
  TYI OFF
  RSXDIR ( QIOW ) ;; IOERR
END

DEFINE ERWNA
        PRINT "ERROR:  WRONG NUMBER OF ARGUMENTS"
END


(* GETFNUM ( REAL , PROMPT )    Commented out for RSX-11M. no floating point
DEFINE GETFNUM REAL
        REAL    FARG1
        INTEGER ARG3
```

```
   GETFNUM := FARG1
   PRINT STR ( ARG3 ) , "[" , #F 4 2 , FARG1 , "]: " , #Z
   IF ( RDLINE )
         IF ( ILITERAL ( LBUF ) )
                GETFNUM := FLOAT ( ILVAL )
         ENDIF
         IF ( RLITERAL ( LBUF ) )
                GETFNUM := RLVAL
         ENDIF
   ENDIF
 END                                                    *)

 (* GETSTRING ( BUFFER , PROMPT )         *)
 DEFINE GETSTRING
        INTEGER ARG1 ARG2
  WHILE ( not WORD )
    PRINT STR ( ARG2 ) , #Z
    RDLINE
  REPEAT
  MVSTR ( TBUF ARG1 )
 END



 DEFINE YESNO INTEGER





        INTEGER ARG1
 LOCAL INTEGER ANSWER
   PRINT STR ( ARG1 ) , " (Y/N): " , #Z
   ANSWER := TYI
   PRINT #A ANSWER
   YESNO := ( ANSWER and 137 ) == ASCII Y

 END


 (* MCONCAT - Concatenates the strings specified as arguments
      MCONCAT dest-string(1st source string) , source-string , .... , arg-count
 *)
 DEFINE MCONCAT COMMAND
        INTEGER STR1
  ITER CMDCNT - 1
        PRINT STR ( STR1 ) , #N
        NXTARG
  LOOP
  NXTARG ( -- ( CMDCNT - 1 ) )
  ENCODE ( STR1 )
 END


 (* IFCR performes a CR if not at the begining of the line *)
 DEFINE IFCR
  IF ( #COLUMN )
```

```
        PRINT
  ENDIF
 END

 (*      Allows for recursion in MAGIC/L.  Calling RECURSE ( Arg1 , Arg2 , ... )
         calls the current subroutine with optional input arguments.      *)
 DEFINE RECURSE IMMFUNC
   CCWD ( . + 1 )
 END

 .MAC

 (*    drop 2 things off the stack    *)
 ENTRY 2DROP
         ADD # 2 , MSP
         NEXT

 (*    divide by 2. done by shifting. 2/ ( -5 ) results in -3 , not -2 as -5 / 2
       in MAGIC  *)
 ENTRY 2/ INTEGER

         ASR (MSP)
         NEXT

 (*  exchange 2 variables. expects pointers as arguments.
     call:   xchg ( ptr ( x ) , ptr ( y ) )       *)
 ENTRY XCHG



         MOV @ 0 (MSP) , R0
         MOV @ 2 (MSP) , R1
         MOV R1 , @ (MSP)+
         MOV R0 , @ (MSP)+
         NEXT


 (*      Move a given number of bytes into the same number of words.
         MVBYWD ( BYTE_ARRAY , BYTE_OFFSET , WORD_ARRAY , #_BYTES )       *)
 entry MVBYWD
         mov     (msp)+ , r0              ; Get number of bytes to transfer.
         mov     (msp)+ , r1              ; Get pointer to word array.
         mov     (msp)+ , r2              ; Get pointer to byte array
         add     (msp)+ , r2              ;  plus the offset.
 0$.     movb    (r2)+ , (r1)+            ; Transfer byte.  Increment pointers.
         clrb    (r1)+                    ; Clear high order byte of word.
         dec     r0                       ; Decrement the count.
         bgt     0$                       ; Branch if count not 0.
         next

 (*      Move a given number of words into the same number of bytes.
         MVBYWD ( BYTE_ARRAY , BYTE_OFFSET , WORD_ARRAY , #_WORDS )       *)
 entry MVWDBY
         mov     (msp)+ , r0              ; Get number of bytes to transfer.
         mov     (msp)+ , r1              ; Get pointer to word array.
```

```
        mov     (msp)+ , r2             ; Get pointer to byte array
        add     (msp)+ , r2             ;  plus the offset.
1$:     movb    (r1)+ , (r2)+           ; Transfer word.  Increment pointers.
        inc     r1                      ; Increment word pointer one more.
        dec     r0                      ; Decrement the count.
        bgt     1$                      ; Branch if count not 0.
        next

.END

INTEGER PAUSECHAR        ; Bucket for read in character
INTEGER SLOW             ; Flag set if slow mode desired
INTEGER FAST             ; Flag set if fast mode desired

(*      PAUSE ROUTINES  *)

DEFINE PAUSE
 IF ( FAST ) RETURN ENDIF
 IFCR
 PRINT "TYPE ANY KEY TO CONTINUE" , #Z
 PAUSECHAR := TYI
 CR
END

DEFINE IFPAUSE




 IF ( SLOW ) PAUSE ENDIF
END

APOP .
```

```
(* The REGION Definition Block *)
record MEM_REC
        integer RGDB ( 0 )        ; Pointer to Region Definition Block.
        integer REGID             ; REGION ID
        integer REGSZ             ; REGION SIZE ( to be set )
        integer REGNM ( 2 )       ; REGION NAME IN RADIX50=(NO NAME)
        integer PARNM ( 2 )       ; NAME OF THE PARTITION IN RADIX50(to be set)
        integer REGST             ; STATUS: USE DEFAULTS(or to be set )
        integer REGPR             ; NOT PROTECTED AT ALL


(* The WINDOW Definition block *)
        integer WNDB ( 0 )        ; Pointer to Window Definition Block
        integer WNDAPR            ; HIGH BYTE HAS THE APR, LOW BYTE IS THE WINDOW
ID
        integer WNDADR            ; VIRTUAL BASE ADDRESS IN TASK'S VIRTUAL SPACE
        integer WNDSZ             ; WINDOW SIZE IN 32WORD BLOCKS
        integer WNDREG            ; REGION ID
        integer WNDOFF            ; OFSSET IN REGION IN 32 WORD BLOCKS
        integer WNDL              ; LENGTH TO MAP IN 32WORD BLOCKS
        integer WNDST             ; WINDOW STATUS WORD
        integer WNDSRB            ; SEN/RECEIVE BUFFER ADDRESS
endrecord

MEM_REC M_MODEL          ; Memory blocks for model access.




MEM_REC M_EDGE           ; Memory blocks for edge image access.

(* Define the memory mngnment executives directives *)

make 'ATRG rsxcall bytewd ( 2 , 57. )
make 'DTRG rsxcall bytewd ( 2 , 59. )
make 'CRAW rsxcall bytewd ( 2 , 117. )
make 'MAPW rsxcall bytewd ( 2 , 121. )
make 'UMAPW rsxcall bytewd ( 2 , 123. )
make 'ELAW rsxcall bytewd ( 2 , 119. )


define AREGION
       integer REGNAM
 ASCR5 ( REGNAM , REGNM ) drop
 REGST := 57K    ; attach it and allow all access
 ATRG ( RGDB )   ; create the region and attache it
 ioerr
end


define DREGION
 DTRG ( RGDB ) ;; ioerr
; if ( REGST )= 40000K ) print "Window unmaped" endif
end
```

```
define cwndow
        integer APR , WNRID , WNSIZ , WNOFF
 WNDAPR := urshift ( APR , 5 )  ; APR in the upper byte
 WNDSZ := WNSIZ          ; MUST BE LESS THAN 4K
 WNDREG := WNRID         ; THE REGION'S ID WHERE THE MAPPING TAKES PLACE
                         ; THIS IS KIND OF TRIKY NO. SO IT MUST BE FETCHED
                         ; FROM THE REGION DEFINITION BLOCK ( RGDB )
 WNDOFF := WNOFF         ; WINDOW OFFSET IN THE REGION
 WNDL off                ; TAKE THE DEFAULT. CAN BE CHECKED FOR THE ACTUAL
                         ; WINDOW SIZE AFTER THE CALL
 WNDST := 202K           ; MAP IT AND ALLOW WRITE ACCESS
 CRAW ( WNDB )  ; CREATE AND MAP THE WINDOW
 IOERR
END


(* This part concerns with random access of a 32kword chunk of memory(region).
The region is "looked at" through a 4kword window which starts at 160000k
absolute address in the Magic task.
        The region can be viewed as 256 x 256 area where each byte corresponds
to a (X,Y) set of coordinates.
        The main access routnes will be:
        - MRDPIX ( X , Y ) for reading a value
        - MWRPIX ( X , Y , VAL ) for writing a value
        Additional routine are provided for setting up the windowing scheme
and filling the region with data from the disk *)

integer YLOW , YHIGH     ; the Y coordinates corresponding to the first and
                         ; last raster in the current window relative to
                         ; whole region.

(* ATTRG ( REGNAM , VADDR ) attachess a named region and an
initial mapping of a 4k window at the begining of the region *)

define ATTRG
        integer REGNAM VADDR
 AREGION ( REGNAM )      ; create a 32kwords dynamic region.
 CWNDOW ( VADDR , REGID , 200K , 0 )
                                 ; create a window at 160000 absolute address
                                 ; of 4k words. Map it at the offset 0 in the
                                 ; region
 YLOW := 0       ; init to the very first raster
 YHIGH := 31.    ; init to 32-nd raster
end


.mac
(*  PIXVAL := MGPIX ( X , YREL ) gives the value of the pixel given the
relative coodinate in the window and the X. *)

entry MGPIX integer
        mov (msp)+ , r1 ; Y-coo
```

```
        mov (msp)+ , r0 ; X-coo
        swab r1 ; Y * 256.
        add r1 , r0      ; relative address from begiinig of the window
        mov @ # ptr ( MEM_REC ) , r2     ; Add active record base address
        add $o WNDADR (r2) , r0
        clr r1
        bisb (r0) , r1  ; get the pixel value
        mov r1 , -(msp) ; return argument
        next


(*  MPPIX ( X , YREL , PIXVAL ) gives the value of the pixel given the
relative coodinate in the window and the X. *)

entry MPPIX
        mov (msp)+ , r2 ; value to be written
        mov (msp)+ , r1 ; Y-coo
        mov (msp)+ , r0 ; X-coo
        swab r1 ; Y * 256.
        add r1 , r0      ; relative address from begiinig of the window
        mov @ # ptr ( MEM_REC ) , r3     ; Add active record base address
        add $o WNDADR (r3) , r0
        movb r2 , (r0)  ; return argument
        next




(*  YREL := REMAP (-Y-COO ) gives the relative coordinates in the region
corresponding to Y-COO. It remaps the window if required. *)

(*
define REMAP integer
        integer YCOO
 if ( not clm ( YCOO , YLOW , YHIGH ) )
        YLOW := lshift ( urshift ( YCOO , 5 ) , 5 )     ; YCOO / 32.*32.
        YHIGH := YLOW + 31.                ; 32 rasters per window
        WNDOFF := YLOW * 4                 ; OFFSE in region is YLOW*256/64
        MAPW ( WNDB )    ; remap the window in the same region
 endif
 REMAP := YCOO - YLOW    ; output YREL
end
*)


entry REMAP integer
        mov     (msp) , r0                 ; Get desired line number.
        cmp     r0 , @ # ptr ( YHIGH )     ; If it is > YHIGH
        bgt     9$                         ;   go to 9$.  (Remap).
        cmp     r0 , @ # ptr ( YLOW )      ; Else If it is >= YLOW
        bge     8$                         ;   go to 8$ (No remap)
9$:     bic     # 37k , r0                 ; Form YLOW.
        mov     r0 , @ # ptr ( YLOW )      ; Store YLOW in YLOW.
        mov     r0 , @ # ptr ( YHIGH )     ; Store YLOW + 31. in YHIGH.
```

SUBSTITUTE SHEET

```
        add     # 31. , @ # ptr ( YHIGH )
        asl     r0                        ; Multiply YLOW by 4.
        asl     r0
        mov     @ # ptr ( MEM_REC ) , r1          ; Active MEM_REC address -> r1.
        mov     r0 , $o WNDOFF (r1)       ; Place YLOW * 4 in current WNDOFF.
        mov     r1 , -(msp)               ; Push active address
        add     # $o WNDAPR , (msp)       ;   + WNDAPR offset. (WNDB pointer).
        mov     # base MAPW , r3          ; Load base address of MAPW routine.
        jsr     pc , xeq                  ; Execute the MAPW (Remap).
8$:     bic     # 177740k , (msp)         ; Return line number - YLOW.
        next

.end

(* PIXVAL := MRDPIX ( XCOO , YCOO ) reads a pixel at XCOO,YCOO *)

define MRDPIX integer
        integer XCOO YCOO
 MRDPIX := MGPIX ( XCOO , REMAP ( YCOO ) )
end


(* MWRPIX ( XCOO , YCOO , PIXVAL ) writes a pixel at XCOO,YCOO *)

define MWRPIX
        integer XCOO YCOO PIXVAL




 MPPIX ( XCOO , REMAP ( YCOO ) , PIXVAL )
end


(* ****************************************************************
This part deals with filling in the region with data from the disk *)

record WND_REC
        integer WNDARR ( 0 )     ; the window is looked at as an array
endrecord

(* FILLREG ( IMFILE ) fills the region with the data provided from the
image file IMFILE.
It assumes previous call to INITRG; i.e. region and fist window mapped *)

define FILLREG
        integer IMFILE
local
        integer IWNDARR IMCH
        char    PNAME ( 30 )
 with M_EDGE
 ATTRG ( "EDGIMG" , 160000k )
 ptr ( WND_REC ) := WNDADR       ; set begining of the array at window virtual
                                 ; address
 mvstr ( "dm3:[5,1]" , PNAME )
 concat ( PNAME , IMFILE )
```

```
 concat ( PNAME , '.IM )
 IMCH := OPEN ( PNAME , 'R )
 REMAP ( 0 )
 iter 8.
  REMAP ( YLOW ) ;; drop
    IWNDARR off
  iter 32.        ; fill in a window
       ; read a raster of 256 bytes from IMFILE into WNDARR at IWNDARR
    rds ( IMCH , PTR ( WNDARR ( IWNDARR ) ) , 256. ) drop
    IWNDARR += 128.      ; next blok
  loop
  YLOW += 32.
 loop
 close ( IMCH )
 DREGION
end


define RECFILL
        integer IMFILE
local
        integer IWNDARR RDBLK IMCH
        char    PNAME ( 30 )
 with M_EDGE
 ATTRG ( "EDGIMG" , 160000k )
 ptr ( WND_REC ) := WNDADR       ; set begining of the array at window virtual
                                 ; address




 mvstr ( "dm3:[100,100]" , PNAME )
 concat ( PNAME , IMFILE )
 concat ( PNAME , '.IM )
 IMCH := OPEN ( PNAME , 'R )
 RDBLK OFF
 REMAP ( 0 )
 iter 8.
  REMAP ( YLOW ) drop
  IWNDARR off
  iter 16.        ; fill in a window
       ; read a raster of 256 bytes from IMFILE into WNDARR at IWNDARR
    rdb ( IMCH , RDBLK , PTR ( WNDARR ( IWNDARR ) ) , 1 ) drop
    INCREMENT RDBLK
    IWNDARR += 256.      ; next blok
  loop
  YLOW += 32.
 loop
 close ( IMCH )
end
```

```
(* --------------------------------------------------------------------
     FLUT.MC -- LOOKUP TABLE SETUP ROUTINES FOR THE LEXIDATA 3400

                  CL        CLRMAP  CS       RGS      COUT     RGOUT
                  SETNIL    RECT    STEP     SETUP    8SETUP   6SETUP

-------------------------------------------------------------------- *)
(*  Variable section: RED, GREEN, and BLUE are the memory locations
    in the Lexidata memory at which the lookup tables start for each
    color.  ALL is a wildcard to effect action for each color.
    TPLANE, GPLANE, and IPLANE are arguments for DSCHAN, the channel
    enabling primitive.
*)

integer RED GREEN BLUE ALL TPLANE GPLANE IPLANE
RED := 1024
GREEN := 2048
BLUE := 3072
ALL on

(*  --------------------------------------------------------------------
    CL -- maps the intensity index (GLIN) into the intensity value (GLOUT)
    to be represented by the Lexidata.  For this command to be meaningful,
    CLIN must be between 0 and 255, 1024 and 1279, 2048 and 2303, or 3072
    and 3327. Only the LSB of GLOUT will be used.
    ==>  GL ( GLIN GLOUT ).
--------------------------------------------------------------------  *)

define CL
        integer GLIN GLOUT
  dsllu ( GLIN GLOUT GLIN GLOUT )
end

(*  --------------------------------------------------------------------
    CLRMAP -- sets every intensity index between 0 and 4095 to the input
    argument LEVEL. There is no harm in setting intensity indices that are
    not used, i.e. out of the range of the memory reserved for each color.
    ==>  CLRMAP ( LEVEL ).
--------------------------------------------------------------------  *)

define CLRMAP
        integer LEVEL
  dsllu ( 0 , LEVEL , 4095 , LEVEL )
end

(*  --------------------------------------------------------------------
    GS -- sets a ramped lookup table with a variety of arguments. GS
    takes as input (1) no arguments (2) 1-3 color names (RED, GREEN, or
    BLUE) or (3) ALL.
    (1)  GS will set up the black-and-white lookup table, from 0 to 255.
    (2)  GS  /RED /GREEN /BLUE  will set up the lookup table starting at
         the appropriate memory location.
    (3)  GS ALL will set up all four lookup tables.
    GS only sets those intensities used by the Lexidata. If in 6-bit mode,
```

```
    only intensities = 0(mod 4) will be set. All others will be set to
    255 (maximum intensity).
    ==)  CS /RED /GREEN /BLUE /ALL
--------------------------------------------------------------------------  *)

define GS command
        integer COLOR
  local
        integer TEMP1 TEMP2
  if ( cmdcnt ==0 )
    dsllu ( 0 , 0 , 255 , 255 )
  else
    TEMP2 := TPLANE + GPLANE + 1
    if ( COLOR == -1 )
      CLRMAP ( 255 )
      iter 4
        TEMP1 := 1024. * I
        iter 256                    ; this sets the lookup tables by
          GL ( TEMP1 + I , I )      ; looping with the right index,
        loop ( TEMP2 )              ; as defined by TPLANE and GPLANE.
      loop
    else
      iter cmdcnt
        dsllu ( COLOR , 255 , COLOR + 255 , 255 )
        iter 256                    ; this sets the lookup tables by
          GL ( COLOR + I , I )      ; looping with the right index,
        loop ( TEMP2 )              ; as defined by TPLANE and GPLANE.




        nxtarg
      loop
    endif
  endif
end

(*  -----------------------------------------------------------------------
    RGS works much the same way as GS does, accepting the same arguments,
    but setting the lookup tables in a downward ramp, i.e., the higher the
    intensity index, the lower the intensity value. If in 6-bit mode, all
    unused indices are set to 0.
    ==)  RGS /RED /GREEN /BLUE /ALL
--------------------------------------------------------------------------  *)

define RGS command
        integer COLOR
  local
        integer TEMP1 TEMP2
  if ( cmdcnt ==0 )
    dsllu ( 0 , 255 , 255 , 0 )
  else
    TEMP2 := TPLANE + GPLANE + 1
    if ( COLOR == ALL )
    CLRMAP ( 0 )
      iter 4
        TEMP1 := 1024 * I
        iter 256
```

```
          GL ( TEMP1 + I , I' )
        loop ( TEMP2 )
      loop
    else
      iter cmdcnt
        iter 256
          GL ( COLOR + I , I' )
        loop ( TEMP2 )
        nxtarg
      loop
    endif
  endif
end

(*  -----------------------------------------------------------------
    GOUT and RGOUT are merely selectors of GS and RCS.
    ==> GOUT
    ==> RGOUT
-------------------------------------------------------------    *)

define GOUT
  GS ALL
end

define RGOUT
  RGS ALL
end




(*  -----------------------------------------------------------------
    SETNIL -- zeroes all used indices within the input color table.
    Used in RECT and STEP to set the indices not chosen to zero. Input
    is a color, or a memory location at which to start setting indices to
    zero. If input is not a color, it must be <= 3739 (256 slots from the
    highest allowable slot). ALL may not be used with SETNIL.
    ==> SETNIL ( /RED /GREEN /BLUE or 0 to 3739 )
-------------------------------------------------------------    *)
define SETNIL command
       integer COLOR
  iter 256
    GL ( COLOR + I , 0 )
  loop ( TPLANE + GPLANE + 1 )
end

(*  -----------------------------------------------------------------
    RECT -- creates a binary lookup table in a certain color, either 0 or
    255 depending on the limits. The first two arguments are the indices
    to set to the maximum and the third is the color table in which to
    work. The first argument must be <= the second argument. ALL may be
    used as a wildcard, but only one color argument is allowed.
    ==> RECT ( 0-255 , 0-255 , /RED       0  LO   HI  1023
                               /GREEN
                               /BLUE         !-----!      255
                               /ALL )     ---!     !----   0
-------------------------------------------------------------    *)
```

SUBSTITUTE SHEET

```
define RECT
        integer LO HI COLOR
   if ( COLOR == -1 )
     CLRMAP ( 0 )
     dsllu ( LO , 255 , HI , 255 )
     dsllu ( RED + LO , 255 , RED + HI , 255 )
     dsllu ( GREEN + LO , 255 , GREEN + HI , 255 )
     dsllu ( BLUE + LO , 255 , BLUE + HI , 255 )
   else
     SETNIL COLOR
     dsllu ( COLOR + LO , 255 , COLOR + HI , 255 )
   endif
 end

(* -----------------------------------------------------------------
    STEP -- a limited version of RECT in which all indices are divided into
    two regions, instead of three. The first input is the index before which
    all values should be zero and after which all values should be set to 255.
    NOTA BENE: the first input is a RELATIVE index, from 0 to 255, not from
    0 to 4095. The second input is a color table within which to make the
    changes. ALL may be used with STEP.
    ==> STEP ( 0-255 , /RED        0    THR.     1023
                       /GREEN
                       /BLUE             !--------       255
                       /ALL )      -----!                0
-----------------------------------------------------------------  *)
define STEP



        integer THRESH COLOR
  if ( COLOR == -1 )
    CLRMAP ( 0 )
    iter 4
     RECT ( THRESH , 255 , 1024 * I )
    loop
  else
    SETNIL COLOR
    RECT ( THRESH , 255 , COLOR )
  endif
end

(* -----------------------------------------------------------------
    8SETUP and 6SETUP set the plane mask variables and call SETUP.
    ==> 8SETUP
    ==> 6SETUP
-----------------------------------------------------------------  *)
define 8SETUP
  TPLANE off            ; if we are coming from 6-bit mode, set TPLANE
  GPLANE off            ; and GPLANE to 0 , and IPLANE to 255, or
  IPLANE := 377K        ; 8 planes enabled for the Lexidata.
  DSCHAN ( TPLANE , GPLANE , IPLANE )
  DSCLR ( TPLANE + GPLANE )
  DSLLU ( 0 , 0 , 255 , 255 )
  DSLLU ( RED , 0 , RED + 255 , 255 )
  DSLLU ( GREEN , 0 , GREEN + 255 , 255 )
```

```
  DSLLU ( BLUE , 0 , BLUE + 255 ., 255 )
end

define 6SETUP
  TPLANE := 1           ; if we are coming from 8-bit mode, set TPLANE
  GPLANE := 2           ; and GPLANE to 1 and 2 respectively, and then
  IPLANE := 374K        ; set IPLANE to 252, or 6 planes enabled for
  DSCHAN ( TPLANE , IPLANE , IPLANE )
  DSCLR ( TPLANE + GPLANE )
  GOUT
end
```

```
(* *****************************************************************
        FXMON - INTERFACES TO THE STANDARD FIRMWARE "IACMON.XB"
***************************************************************** *)
APUSH RADIX
OCTAL

INTEGER LEXIOFLG                ; FLAG TO TELL WBUSY WHETHER TO WAIT FOR OUTPUT
                                ; OR INPUT

(* *****************************************************************
        DMA TRANSFER ROUTINES (COMMAND 10)
***************************************************************** *)
(* DEFINE DMACT ACTION
        INTEGER ARG1 ARG2 ARG3 ARG4 ARG5
        INTEGER OCODE ( 1 )
 WDOAS ( OCODE ( 0 ) )
 WDOAS ( DCHAD ( ARG1 ) ) WDOAS ( ARG2 ) WDOAS ( -- ARG3 )
 WDOAS ( BYTEWD ( ARG5 , ARG4 ) )
END  *)

(* WBUSY -- WAIT FOR INPUT/OUTPUT FROM LEXIDATA     *)

DEFINE WBUSY
        IF ( LEXIOFLG ) DSOWT
        ELSE DSIWT
        ENDIF
END
```

```
(*  DMAW -- WRITE SEQUENTIAL PIXELS WITHIN AN AREA. WORD MODE
    CALL:   DMAW ( BUFFER , X0 , XL , Y0 , YL )         *)

DEFINE DMAW
  INTEGER BUFFER ( 1 ) AX0 AXL AY0 AYL
        DSLIM ( AX0 , AY0 , AX0 + AXL - 1 , AY0 + AYL - 1 )
        DSPUT ( BUFFER , AXL * AYL )
        LEXIOFLG ON
END

(*  DMAR -- READ SEQUENTIAL PIXELS FROM DISPLAY INTO BUFFER. WORD MODE
    CALL:   DMAR ( BUFFER , X0 , XL , Y0 , YL )         *)

DEFINE DMAR
  INTEGER BUFFER ( 1 ) AX0 AXL AY0 AYL
        DSLIM ( AX0 , AY0 , AX0 + AXL - 1 , AY0 + AYL - 1 )
        DSGET ( BUFFER , AXL * AYL )
        LEXIOFLG OFF
END

DEFINE PDMAR
        INTEGER BUFFER AX0 AXL AY0 AYL
END

DEFINE PDMAW
        INTEGER BUFFER AX0 AXL AY0 AYL





END

(* WRPIX - WRITE A SINGLE PIXEL
CALL:        WRPIX ( IX , IY , LEVEL )          *)

DEFINE WRPIX
        INTEGER IX IY LEVEL
LOCAL INTEGER ARR ( 3 )
        ARR ( 0 ) := IX ;; ARR ( 1 ) := IY ;; ARR ( 2 ) := LEVEL
        DSRNW ( 1 , ARR )
END

(* RDPIX - READ A SINGLE PIXEL
CALL:        LEVEL := RDPIX ( IX , IY )         *)

DEFINE RDPIX INTEGER
        INTEGER IX IY
LOCAL INTEGER ARR ( 2 )
        ARR ( 0 ) := IX ;; ARR ( 1 ) := IY
        DSRNR ( 1 , ARR , PTR ( RDPIX ) )
END

(* GBAR - FILL A BLOCK OF PIXELS
CALL:     GBAR ( X0 , XL , Y0 , YL , LEVEL )     *)

DEFINE GBAR
        INTEGER AX0 AXL AY0 AYL LEVEL
```

```
   IF ( AXL ==0 ) RETURN ENDIF
     DO AY0 , MIN ( AY0 + AYL - 1 , 511. )
          DSVEC ( AX0 , I , MIN ( AX0 + AXL - 1 , 639. ) , I , LEVEL )
   LOOP
   END

DEFINE BLKSUM LONG
        INTEGER ARG1 ARG2 ARG3 ARG4 ARG5
END

(*      Define DELAY function using RSX Mark Time directive and Wait
        for Global Event Flag directive.
make    'MRKT$  rsxcall bytewd ( 5 , 23. )
make    'STSE$  rsxcall bytewd ( 2 , 135. )
*)

integer MRKT$ ( 0 )
.word   bytewd ( 5 , 23. )
.word   23.
.blkw   1
.word   1
.word   0

integer WTSE$ ( 0 )
.word   bytewd ( 2 , 41. )
.word   23.



define DELAY
        integer DTIM
  MRKT$ ( 2 ) := DTIM * 6
  RSXDIR ( MRKT$ ) ioerr
  RSXDIR ( WTSE$ ) ioerr
end


APOP
```

```
apush radix
octal
(*

        Subroutne 22BADDR does a conversion of the 16-bit virtual address
        supplied as argument into a full 22-bit physical address of the Q-bus.
        The MMU user map registers are used for this purpose so this subroutine
        must be used in a magic/l environment linked to the I/O page.
        For how to link to the I/O page see MGLIOP.CMD file.
                input: 16 bit word representing the virtual address
                output: long(32-bit) word representing the 22-bit address
                        as following:
                        - ls part is the low 16-bits
                        - ms part is the high 6 bits multiplied by 2.
                note: this format was chosen to correspond to the CCD CAMERA
                      CONTROLLER build by Zvi Orbach. More bit manipulations
                      might be required if used with other devices.
                calling sequence:
                        long 22bitaddrs
                        22bitaddr := 22baddr ( ptr ( buff ) )
                                                                    *)
.mac
; the user map register addresses in i/o page
label   UPAR ;; .word   177640k
label   UPDR ;; .word   177600k

entry   22ADDR  long


        mov     (msp) , r0      ; get the virtual address
        mov     r0 , r1         ;
        rol     r0              ; isolate the APF ( Active Page Field )
        rol     r0
        rol     r0
        rol     r0              ;
        bic     # 177770k , r0  ; in r0
        asl     r0              ; even
        bic     # 160000k , r1  ; isolate DF ( Displacemant Field ) in r1
        mov     r1 , r3         ; save it
        bic     # 177700k , r1  ; isolate the displacemant in block
        ash     # 177772 , r3   ; block # in page
        add     UPAR , r0       ; get the corresponding PAR addr in i/o page
        add     (r0) , r3       ; 16 bits physical address in blocks
        clr     r2              ; ( r2,r3 ) will be the 22 bit physical addr
        ashc    # 6 , r2                ; make place for the additional 6 bits
        add     r1 , r3         ; finally... the 22 bit address
        mov     r3 , (msp)      ; least significant portion of the address
        asl     r2              ; CCD camera controller format
        mov     r2 , -(msp)     ; push the extra 6 bits in the stack
        next
.end

apop
```

```
parameter WCR := 172410k          ; DMA word count register.
parameter BAR := 172412k          ; Bus address register for DMA.
parameter CSR := 172414k          ; Control status register.
parameter DBR := 172416k          ; Data buffer register.

long    PHYADR                    ; Physical (22-bit) address of the buffer.

record DMALINE
        integer LNBUF ( 256. )
endrecord

DMALINE INLN ( 2 )
integer OUTLN ( 256. )
char    PNAME ( 30. )

integer TX0 TY0
integer IFN ( 10 ) IDPN ( 20 )
integer VCH
integer CFLAG
CFLAG off


(*      Wait until DMA operation is complete.  (Monitors BUSY bit.)     *)
define WBUS?Y




  while ( peek ( CSR ) AND 200k )
  repeat
end

define RDLN
        integer BUFF ( 1 ) x0 x1 y0 y1
  PHYADR := 22ADDR ( BUFF )
  poke ( 130000k + X0 - 1 , DBR )
  poke ( 114000k + Y0 , DBR )
  poke ( -- XL / 2 , WCR )
;  poke ( -- ( XL * YL ) , WCR )
  poke ( lsword ( PHYADR ) , BAR )
  poke ( 0 , DBR )
  poke ( msword ( PHYADR ) + 1 , CSR )
end

(*
define WRTLN
        integer X0 Y0 LEN
  if ( CFLAG ==0 )
        poke ( 130000k + X0 , DBR )
        poke ( 134000k + Y0 , DBR )
  else
        poke ( 132000k + X0 , DBR )
        poke ( 136000k + Y0 , DBR )
  endif
  iter LEN
```

```
        poke ( ( OUTLN ( i ) and 777k ) + 120000k , DBR )
    loop
  end
  *)

  mvstr ( 'dm3:[5,1] , IDPN )

  define IMAGEFN
        integer ARG1
    mvstr ( ARG1 , IFN )
    mvstr ( IDPN , PNAME )
    mconcat  PNAME , ARG1 , '.im , 3
    print str ( PNAME )
  end


  define VDRAW
        integer FNAME X0 Y0
   local
        integer BUFPTR
    IMAGEFN ( FNAME )
    VCH := open ( PNAME , 'r )
    BUFPTR off
    do Y0 , Y0 + 255.
      if ( rds ( VCH , OUTLN , 256. ) <> 256. )
        print "WARNING:  Unexpected end of file"
        exit
      else
        MVBYWD ( OUTLN , 0 , INLN ( BUFPTR ) , 256. )
        DMAW ( INLN ( BUFPTR ) , X0 , 256. , i , 1 )
        BUFPTR := 1 - BUFPTR
      endif
    loop
    WBUSY
    close ( VCH )
  end


  define REDRAW
    VDRAW ( IFN )
  end

  define VSAVE
        integer FNAME
   local
        integer BUFF1   BUFF2
    with M_EDGE
      ATTRC ( "IOPAGE" , 160000K )
    IMAGEFN ( FNAME )
    VCH := open ( PNAME , 'rwct )
    BUFF1 := ptr ( INLN ( 0 ) )
    BUFF2 := ptr ( INLN ( 1 ) )
    RDLN ( BUFF1 , 128. , 256. , 128. , 1 )
    do 129. , 384.
```

```
        WBUS?Y
        RDLN ( BUFF2 , 128. , 256. , 1 , 1 )
        WRS ( VCH , BUFF1 , 256. )
        XCHG ( ptr ( BUFF1 ) , ptr ( BUFF2 ) )
  loop
  WBUS?Y
  DREGION
  close ( VCH )
end

define GRABIM
        integer X0 Y0
 local
        integer BUFPTR
  with M_EDGE
    ATTRG ( "IOPAGE" , 160000K )
  BUFPTR off
  poke ( 1000k , DBR )
  delay ( 1 )
  poke ( 0 , DBR )
  do 128. 383.
       WBUS?Y
       RDLN ( INLN ( BUFPTR ) , 128. , 256. , 1 , 1 )
       MVBYWD ( INLN ( BUFPTR ) , 0 , OUTLN , 256. )
       DMAW ( OUTLN , X0 , 256. , Y0 + 1 - 128. , 1 )
       BUFPTR := 1 - BUFPTR



  loop
  WBUSY
  DREGION
end
```

```
(*

integer BLKBUF ( 0 )
.word   39.
.word   1 - 256. / 2
.blkw   128.

define PDMAW
        integer BUFFER ( 1 )
  mvwds ( BUFFER , BLKBUF + 4 , 128. )
  DSOWT
  DSBLOC ( BLKBUF , 130. )
end

*)

define $DRAW
        integer TX0     TY0
 local
        integer IWNDARR LNCNT
  DSLIM ( TX0 , TY0 , TX0 + 255. , TY0 + 255. )
  YLOW := 32.
  iter 256.
    REMAP ( i ) drop
    IWNDARR := WNDADR
    iter 32.
        MVBYWD ( IWNDARR , 0 , OUTLN , 256. )




        DSPUT ( OUTLN , 256. ) VBUSY
        IWNDARR += 256.
    loop
  loop ( 32. )
  REMAP ( 0 ) drop
end


define DRAW
 with M_EDGE
  ATTRG ( "EDGIMG" , 160000k )
  $DRAW
  DREGION
end


define MDRAW
 with M_EDGE
  ATTRG ( "MTCHIM" , 160000k )
  $DRAW
  DREGION
end


define MSAVE
        integer FNAME
 local
```

```
        integer OUTCH IWNDARR PNAME ( 15. )
    mvstr ( "dm3:[200,200]" , PNAME )
    concat ( PNAME , FNAME )
    concat ( PNAME , '.im )
    OUTCH := open ( PNAME , 'rwct )
   with M_EDGE
    ATTRG ( "EDGIMG" , 160000k )
    YLOW := 32.
    iter 256
     REMAP ( 1 ) drop
     IWNDARR := WNDADR
     iter 32.
         WRS ( OUTCH , IWNDARR , 256. )
         IWNDARR += 256.
     loop
    loop ( 32. )
    REMAP ( 0 ) drop
    DREGION
    close ( OUTCH )
end
```

```
(* Global event flags for synchronisation. TASK holds the name of the task
   with whom we are communicating.  THESE MUST BE GLOBAL    *)

integer SYNC1 SYNC2 TASK ( 2 )

(* Define executive directives to be used for tasking with Control *)

make 'WAIT rsxcall bytewd ( 2 , 41. )

make 'CLEAR rsxcall bytewd ( 2 , 31. )

make 'READ rsxcall bytewd ( 2 , 39. )

make 'SET rsxcall bytewd ( 2 , 33. )

make 'RCVD$ rsxcall bytewd ( 4 , 75. )

make 'SDAT$ rsxcall bytewd ( 5 , 71. )

(*  Execute a subroutine call. the arguments and subroutine name are in BUFF.
BUFF contains. TASK1 , TASK2 , 0 or -2 , #ARGS , arg1 , arg2 , .. argn , subrout
ine
    TASK1 and 2 make up the taskname of the caller  *)

define DOROUTINE
        integer BUFF ( 1 )
```

```
local
        integer ADR OFFST OFFST1
  OFFST := 4
  OFFST1 := 0
  if ( BUFF ( 2 ) )
    ptr ( BUFF ( 4 ) )
    OFFST1 := 1
    OFFST := OFFST + length ( ptr ( BUFF ( 4 ) ) ) / 2 + 1
  endif
  lookup ( ptr ( BUFF ( BUFF ( 3 ) + OFFST - OFFST1 ) ) ) ;; ADR := lastword
  DROP
  iter BUFF ( 3 ) - OFFST1
    ( BUFF ( I + OFFST ) )                          ; store args on stack
  loop
  exec ( ADR )
end

(* Send a buffer of data to the task we are connected to. BUFF ( 0 ) must
   be greater than 0. This can be used in the receiver as a code for what
   data has been sent. The buffer can be no longer than 13 words.  *)

define SEND
        integer BUFF ( 1 )
  SDAT$ ( TASK ( 0 ) , TASK ( 1 ) , BUFF , SYNC1 ) ;; ioerr
  WAIT ( SYNC2 )
  CLEAR ( SYNC2 )
end




(* Receive data from that task we are connected to and put it in a buffer
  Call:   RECEIVE ( BUFFER )  Note: if these routines are overlaid , BUFFER
  must be global.  The buffer must be at least 15 words. BUFFER contains:
     TASK1 , TASK2 , CODE , DATA      where TASK1 AND 2 make the name of the
  task which is sending the message. CODE is 0 if we are calling a routine,
  -1 if the other task is informing us of its rundown, and )0 if other data
  has been sent                                   *)

define RECEIVE
        integer BUFF ( 1 )
  WAIT ( SYNC1 )
  CLEAR ( SYNC1 )
  RCVD$ ( 0 , 0 , BUFF ) ;; ioerr
  if ( BUFF ( 2 ) == -1 )                               ; rundown
    SET ( SYNC2 )                                       ; acknowledge receipt
    if ( BUFF ( 3 ) ==0 ) bye else return endif
  else
    if ( BUFF ( 2 ) ==0 or BUFF ( 2 ) == -2 )
        DOROUTINE ( BUFF )
    endif
    SET ( SYNC2 )
  endif
end

(* Set up communications with the task that requested us.
   receives the two flags that the tasks will use for synchronization.
```

```
   and the name of the task with whom we are communicated.
   Any task that gets connected to must issue an INITREC command before
   proceeding.        *)

define INITREC
local
        integer BUFF ( 15 )
  detterm
  RCVDs ( 0 , 0 , BUFF ) ;; ioerr
  iter 2
    TASK ( I ) := BUFF ( i )
  loop
  SYNC1 := BUFF ( 2 )
  SYNC2 := BUFF ( 3 )
  SET ( SYNC2 )
end
```

```
EXT DISPMODEL
EXT VFMP
EXT BLBDISP                            ; blob bounding rectangles
EXT HAROLD                             ; &&&
integer STPFLAG

integer VIDCBF ( 15. )

define CONNECT_2_MASTER
 INITREC
 begin
        RECEIVE ( VIDCBF )
 until ( STPFLAG )
end


define RECONNECT
  SET ( SYNC2 )
  begin
        RECEIVE ( VIDCBF )
  until ( STPFLAG )
end

integer TMPICH  TMPOCH
```

```
define STOPCO
        integer TERM
  TMPICH := cich
  TMPOCH := cooh
  cich := open ( TERM , 'rwa )
  coch := cich
  poke ( 2 , fdb ( coch ) )
  atterm
  STPFLAG on
end

define STRTCO
  detterm
  close ( cich )
  cich := TMPICH
  coch := TMPOCH
  STPFLAG off
  RECONNECT
end

define INITVID
  INITD
  CONNECT_2_MASTER
end



$RESTART := base INITVID        ; entry point for VIDEOT.TSK

SAVE WFVIDEOT
```

```
parameter MAX_#_ENT := 20        ; Maximum # of permissible entities.
parameter #POINTS := 25          ; Maximum # of points permitted within
                                 ;  an entity.

record POINT_REC
        integer XI YI            ; Coordinates of first corner point.
        integer CURTYPE          ; Type of the line between 1st and 2nd point.
        integer XJ YJ            ; Coordinates of second point.
        integer NXTTYPE          ; Type of the next line (between 2nd and 3rd).
        dummy -3
endrecord

record ENTITY
        integer #PTS                     ; # of points.
        POINT_REC ZI ( #POINTS )         ; See record POINTS.
endrecord

record FRAME_REC
        integer FRM#             ; Frame # .
        integer #ENT             ; # of entities.
        ENTITY EI ( MAX_#_ENT )
        integer HX1     HX2     HY1     HY2      ; Horizontal landmark points.
        integer VX1     VX2     VY1     VY2      ; Vertical landmark points.
        integer HX3     HX4     HY3     HY4      ; Horizontal landmark points.




        integer VX3     VX4     VY3     VY4      ; Vertical landmark points.
endrecord

define DISPMODEL
        integer TX0 TY0
  GBAR ( TX0 , 256. , TY0 , 256. , GRNGL )
  with M_EDGE
    ATTRG ( "MODELR" , 160000k )
    ptr ( FRAME_REC ) := WNDADR
    iter #ENT
      with EI ( i )
        iter #PTS - 1
          with ZI ( i )
                DSVEC ( XI + TX0 , YI + TY0 , XJ + TX0 , YJ + TY0 , REDGL )
        loop
    loop
    DSVEC ( HX1 + TX0 , HY1 + TY0 , HX2 + TX0 , HY2 + TY0 , REDGL )
    DSVEC ( VX1 + TX0 , VY1 + TY0 , VX2 + TX0 , VY2 + TY0 , REDGL )
    DSVEC ( HX3 + TX0 , HY3 + TY0 , HX4 + TX0 , HY4 + TY0 , REDGL )
    DSVEC ( VX3 + TX0 , VY3 + TY0 , VX4 + TX0 , VY4 + TY0 , REDGL )
    DREGION
end
```

```
(* Display the Wafer Map module *)

(* Attachement to the Inspection Plan *)

MEM_REC M_IPSDB

define ATIPSDB
 with M_IPSDB
 ATTRG ( "IPSDBR" , 160000K )
 ptr ( IPSDB_REC ) := WNDADR
end

integer MPX0 , MPY0      ; the origin x,y of the wafer map on the screen
integer MPRAD            ; the map size on the screen
real    MPSCL            ; # pixel/micron
integer XPI YPI          ; the pitches
integer STH AVW          ; str and ave sizes
integer DIH DIW          ; die sizes
integer XROW YROW        ; current row to display
integer GLDIE            ; the gray level at which the die boundaries are display
ed
integer GLCIR            ; the gray level of the circle
integer CROSX ( MAX_RETICLES )  ; x location of the marker for the die being ins
pected
integer CROSY ( MAX_RETICLES )  ; y location of the marker for the die being ins

pected

(* Get the current row coordinates *)
define GROWCOO
       integer ROW COL
 XROW := MPX0 + COL * XPI
 YROW := MPY0 + ROW * YPI
end

define GTORG
       integer X0 Y0 SZ
 with FLAT_TO_ORIGIN
 MPX0 := X0 + ( SZ - fix ( float ( X ) * MPSCL ) )
 MPY0 := Y0 + fix ( float ( Y ) * MPSCL )
end

define BOX
       integer XDI , YRO , GL
    DSVEC ( XDI , YRO , XDI + DIW , YRO , GL )
    DSVEC ( XDI , YRO + DIH , XDI + DIW , YRO + DIH , GL )
    DSVEC ( XDI , YRO , XDI , YRO + DIH , GL )
    DSVEC ( XDI + DIW , YRO , XDI + DIW , YRO + DIH , GL )
end

define WFMAP
       integer X0 , Y0 , SZ
local
```

```
        integer XDIE

 ATIPSDB

 GLDIE := GRNGL
 GLCIR := GRNGL

 with INSP_PLN
 with HEADER
 with INSP_DATA_BASE
 SZ -= 10.
 X0 += 5
 Y0 += 5
 MPRAD := SZ / 2
 MPSCL := float ( MPRAD ) / ( float ( WAFER_SZ ) * 500.0 )

 XPI := fix ( DIE_X * MPSCL )
 YPI := fix ( DIE_Y * MPSCL )

 with LAYERS ( CUR_LAYER )
 with DTL_LAYER_REV ( #_REVS - 1 )
 with L_RETICLE
 with RETICLE_DIE
 STH := D_ST_HGT
 AVW := D_AV_WDTH
 DIW := YPI - fix ( float ( AVW ) * MPSCL )
 DIH := XPI - fix ( float ( STH ) * MPSCL )




 GTORG ( X0 Y0 MPRAD )
 iter #_DIE_ROWS
   with WAFER_MAP ( i )
   GROWCOO ( i , 1ST_D_# )
   XDIE := XROW
   iter LAST_D_# - 1ST_D_#
     BOX ( XDIE , YROW , GLDIE )
     XDIE := XDIE + XPI
   loop
 loop
 DSCIR ( ( X0 + MPRAD ) , ( Y0 + MPRAD ) , MPRAD , GLCIR )
 with REFERENCE_DIE
 GROWCOO ( ROW , CLMN )                   ; GET COORDINATES OF REFERENCE_DIE
 BOX ( XROW , YROW , BLUGL )              ; DRAW IT IN RED
 with L_INSPECTION                        ; GET RETICLES TO INSPECT
 iter #_TO_INSP
    with INSP_R ( i )
    GROWCOO ( ROW , CLMN )
    CROSX ( i ) := XROW + DIW / 2 - 2
    CROSY ( i ) := YROW + DIH / 2 - 3
    BOX ( XROW , YROW , REDGL )
 loop
 DREGION
end

define DISPX
```

```
    integer X0 Y0 CL
    DSSAO ( X0 Y0 CL 0 1 )
    DSTXT ( "X" )
  end

  define SHOWDIE
      ATIPSDB
      if ( I-MODE == PRIMARY )
        DISPX ( CROSX ( CONFIRM ) , CROSY ( CONFIRM ) , 0 )
        DISPX ( CROSX ( PRIMARY ) , CROSY ( PRIMARY ) , REDGL )
      else
        DISPX ( CROSX ( PRIMARY ) , CROSY ( PRIMARY ) , 0 )
        DISPX ( CROSX ( CONFIRM ) , CROSY ( CONFIRM ) , REDGL )
      endif
      DREGION
  end

  define DMESSAGE
    address MESS
    CBAR ( 0 288. 256. 90. 0 )
    DSSAO ( 32. 320. REDGL 0 2 )
    DSTXT ( MESS )
  end

  define ACQMSG
    DMESSAGE ( "IMAGE ACQUISITION" )
  end




  define IPRMSG
    DMESSAGE ( "IMAGE PROCESSING" )
  end

  define MDLMSG
    DMESSAGE ( "MODEL MATCHING" )
  end

  define DFTMSG
    DMESSAGE ( "DEFECT ANALYSIS" )
  end

  define ENDMSG
    DMESSAGE ( "THAT'S ALL FOLKS!" )
  end
```

```
integer MINX0 MINY0 MAXX1 MAXY1

define BOUND
  DSVEC ( MINX0 - 1 , MINY0 - 1 , MAXX1 + 1 , MINY0 - 1 , REDGL )
  DSVEC ( MAXX1 + 1 , MINY0 - 1 , MAXX1 + 1 , MAXY1 + 1 , REDGL )
  DSVEC ( MAXX1 + 1 , MAXY1 + 1 , MINX0 - 1 , MAXY1 + 1 , REDGL )
  DSVEC ( MINX0 - 1 , MAXY1 + 1 , MINX0 - 1 , MINY0 - 1 , REDGL )
end

define BNDRCTS
        integer TX0 TY0
  ATIPSDB
  with INSP_DATA_BASE
   TX0 += REG_X
   TY0 += REG_Y
  with INSP_PLN
   with LAYERS ( MOD_LAYER )
    with DTL_LAYER_REV ( #_REVS - 1 )
     with L_RETICLE
      with RETICLE_DIE
       with D_PATTERNS ( MOD_PATTERN )
        with INSP_FR ( MOD_SITE )
         with F_DEFCTS ( MOD_FRAME )
        print "No. of Defects" , #_DFCTS
           iter #_DFCTS
             with DEFECTS ( i )
              MINX0 := XCOM - DELX + TX0




              MINY0 := YCOM - DELY + TY0
              MAXX1 := XCOM + DELX + TX0
              MAXY1 := YCOM + DELY + TY0
              BEEP
              BOUND
              DELAY ( 5 )
           loop
  DREGION
end
```

```
define DRAWBOX
        integer X0 XL Y0 YL GL
  local
        integer X1 Y1
  X1 := X0 + XL                        ; sets up the new coordinates of the
  Y1 := Y0 + YL                        ; right bottom
  dsvec ( X0 , Y0 , X1 , Y0 , GL )    ; draws the new box according to specs
  dsvec ( X1 , Y0 , X1 , Y1 , GL )
  dsvec ( X1 , Y1 , X0 , Y1 , GL )
  dsvec ( X0 , Y1 , X0 , Y0 , GL )
end

define LOWMAG
  DSCLR ( 255. )
  VDRAW ( 'JOE 32. 256. )
  iter 6
  DRAWBOX ( 201. , 24 , 346. + ( I * 24 ) , 24 , REDGL )
  LOOP
  PAUSE
end

define LMAG
  ATIPSDB
  GBAR ( 520. , 30. 345 , 150 0 )
  DRAWBOX ( 521. , 24 , 346. + ( ( 5 - CUR_FRAME ) * 24 ) , 24 , REDGL )
  DREGION
end




define HAROLD_DEMO
  VDRAW ( 'RNF03 32. 0 )
  DRAWBOX ( 47. , 226. , 15. , 226. REDGL )
  PAUSE
  DISPMODEL ( 352. 0 )
  PAUSE
  VDRAW ( 'RNF03 352. 256. )
  DRAWBOX ( 547. 17. 308. 18. REDGL )
  DRAWBOX ( 428. 17. 357. 15. REDGL )
  PAUSE
  DSCLR ( 255. )
  WFMAP ( 220. 56. 200 )
  VDRAW ( 'RNF03 32. 256. )
  VDRAW ( 'RCL03 352. 256. )
  DSVEC ( 318. 155. 32. 254. REDGL )
  DSVEC ( 318. 155. 288. 254. REDGL )
  DSVEC ( 348. 155. 352. 254. REDGL )
  DSVEC ( 348. 155. 608. 254. REDGL )
  DRAWBOX ( 227. 17. 308. 18. REDGL )
  DRAWBOX ( 547. 17. 308. 18. REDGL )
end
```

EDGE DETECTION

```
(* ***********************************************************************
        CEDGET.MG - THIS MODULE LOADS ALL OF THE MODULES USED IN "CEDGET"
 *********************************************************************** *)

ext     PDPID
ext     DMISC
ext     MAKAP
ext     EDGREG
ext     APDECL
ext     INITAP
ext     APEDGE
ext     EDGCOM

integer STPFLAG
integer EDGCBF ( 15. )

define CONNECT_2_MASTER
 INITREC
 begin
        RECEIVE ( EDGCBF )
 until ( STPFLAG )
end

define RECONNECT
  SET ( SYNC2 )
  begin
        RECEIVE ( EDGCBF )
  until ( STPFLAG )
end

integer TMPICH  TMPOCH

define STOPCO
        integer TERM
  TMPICH := cich
  TMPOCH := coch
  cich := open ( TERM , 'rwa )
  coch := cich
  poke ( 2 , fdb ( coch ) )
  atterm
  STPFLAG on
end

define STRTCO
  detterm
  close ( cich )
  cich := TMPICH
  coch := TMPOCH
```

```
  STPFLAG off
  RECONNECT
 end

define EDGEINIT
  with M_EDGE
    ATTRG ( "EDGIMG" , 140000k )
  CONNECT_2_MASTER
end

mvstr ( 'cedget , promstr )

frestart := base EDGEINIT        ; RESTART FOR EDGE DETECTION PROGRAM

save CEDGET
```

```
(*      Name : APEDGE.M                              *)

integer COUNT

define INIT_DBF
 iter 40
        DBF ( I ) := I + 1
 loop
end


define FIRST
        ACHN ( ptr ( N ) , FCBCHN , 500. )      ; Perform 1,2,1
        AADD ( DBF ( 26 ) , DBF ( 18 ) , DBF ( 18 ) ) ; hor. conv
        AADD ( DBF ( 26 ) , DBF ( 26 ) , DBF ( 17 ) ) ; --> DBF ( 26 )
        AADD ( DBF ( 26 ) , DBF ( 26 ) , DBF ( 19 ) ) ;

        ACONV ( DBF ( 27 ) , DBF ( 26 ) , 1 )          ; HO --> DBF ( 27 )
        ASCDB ( -2. DBF ( 26 ) )
        ACONV ( DBF ( 28 ) , DBF ( 26 ) , 2 )          ; HE --> DBF ( 28 )

        AHORZ ( DBF ( 32 ) , DBF ( 27 ) , DBF ( 28 ) ) ; HEDGE ( CUR_LINE )

        ASUB ( DBF ( 26 ) , DBF ( 16 ) , DBF ( 20 ) ) ;
        AMULS ( DBF ( 26 ) , DBF ( 26 ) , 1 )          ; Equivalent
        ASUB ( DBF ( 29 ) , DBF ( 17 ) , DBF ( 19 ) ) ; vert. conv.
```

```
        AMULS ( DBF ( 29 ) , DBF ( 29 ) , 3 )          ; with odd mask.
        AADD ( DBF ( 30 ) , DBF ( 29 ) , DBF ( 26 ) ) ; VO --> DBF ( 29 )

        ATFR1 ( DBF ( 29 ) DBF ( 30 ) 7 )
        ASCDB ( 1 DBF ( 29 ) )

        AADD ( DBF ( 26 ) , DBF ( 16 ) , DBF ( 20 ) ) ;
        AMULS ( DBF ( 26 ) , DBF ( 26 ) , 2 )          ; Equivalent
        AADD ( DBF ( 30 ) , DBF ( 17 ) , DBF ( 19 ) ) ; vert. conv.
        AMULS ( DBF ( 30 ) , DBF ( 30 ) , 5 )          ; with even mask.
        AADD ( DBF ( 30 ) , DBF ( 30 ) , DBF ( 26 ) ) ; VE --> DBF ( 30 )
        AMULS ( DBF ( 26 ) , DBF ( 18 ) , 6 )          ;
        AADD ( DBF ( 26 ) , DBF ( 30 ) , DBF ( 26 ) ) ;

        ASCDB ( -1 DBF ( 26 ) )
        ATFR1 ( DBF ( 30 ) DBF ( 26 ) 7 )

        ACEND
        AHIAB ( FCBCHN , 8. , N )
        AXCHN ( 8. )
        ARLDB ( 8. )
end

define ROT_SCRATCH
 local
        integer TEMP1 TEMP2 TEMP3 TEMP4
 TEMP2 := DBF ( 20 )




 do 16 , 19
        TEMP1 := DBF ( I' )
        DBF ( I' ) := TEMP2
        TEMP2 := TEMP1
 loop
        DBF ( 20 ) := TEMP2
end


define ROT_OUT
 local
        integer TEMP1 TEMP2
 TEMP2 := DBF ( 9 )
 do 10 , 15
        TEMP1 := DBF ( I )
        DBF ( I ) := TEMP2
        TEMP2 := TEMP1
 loop
        DBF ( 9 ) := TEMP2
end

define UPDATE
 ROT_SCRATCH
 ACHN ( ptr ( N ) , FCBCHN , 500. )
 AHIAB ( BOT , DBF ( 31 ) , 128. )
 AUPAK ( DBF ( 20 ) , DBF ( 31 ) )
```

```
 ADNSN ( DBF ( 20 ) )
 ANRDB ( DBF ( 20 ) )
 ADBDB ( DBF ( 36 ) DBF ( 29 ) )
 ADBDB ( DBF ( 37 ) DBF ( 30 ) )
 ADBDB ( DBF ( 9 ) DBF ( 32 ) )
 AZRDB ( DBF ( 34 ) )
 ADBDB ( DBF ( 34 ) DBF ( 35 ) )
 AZRDB ( DBF ( 35 ) )
 ACEND
 AHIAB ( FCBCHN , 8. , N )
 AXCHN ( 8. )
 ARLDB ( 8. )
end


define DOLINES
        integer #LINES
 iter #LINES
        FIRST
        increment COUNT
        if ( COUNT < 2 )
               UPDATE
        else
               AVZER ( DBF ( 36 ) DBF ( 37 ) DBF ( 29 ) DBF ( 30 ) ^
                      DBF ( 34 ) DBF ( 35 ) )



               AORDB ( DBF ( 9 ) DBF ( 34 ) )
               ROT_OUT
               if ( COUNT < 4 )
                      UPDATE
               else
        AFRUN ( DBF ( 9 ) , DBF ( 10 ) , DBF ( 11 ) , DBF ( 12 ) , ^
                      DBF ( 13 ) DBF ( 14 ) , DBF ( 15 ) )
               if ( COUNT < 7 )
                      UPDATE
               else
                      AABHI ( TOP , DBF ( 15 ) , 256. , 0 )
                      MVWDBY ( TOP , 0 , TOP , 256. )
                      UPDATE
               endif
               endif
        endif
        ptr ( LINE_REC ) += 256.
 loop
end


define DOEDGE
  COUNT off
  INIT_DBF                         ; Initilize AP DBF values
  ZERO_DBF
  READ_INIT
```

```
mvzer ( WNDADR , 256. )
DOLINES ( 24. )
do 64. , 832.
      WNDOFF := i
      MAPW ( WNDB )
      ptr ( LINE_REC ) := WNDADR + 1024.
      DOLINES ( 16. )
loop ( 64. )
WNDOFF := 896.
MAPW ( WNDB )
ptr ( LINE_REC ) := WNDADR + 1024.
DOLINES ( 19. )
iter 4
      ROT_OUT
      APRUN ( DBF ( 9 ) , DBF ( 10 ) , DBF ( 11 ) , DBF ( 12 ) , ^
                       DBF ( 13 ) DBF ( 14 ) , DBF ( 15 ) )
      AABHI ( TOP , DBF ( 15 ) , 256. , 0 )
      MVWDBY ( TOP , 0 , TOP , 256. )
      ptr ( LINE_REC ) += 256.
loop
iter 3
      ROT_OUT
      AABHI ( TOP , DBF ( 15 ) , 256. , 0 )
      MVWDBY ( TOP , 0 , TOP , 256. )
      ptr ( LINE_REC ) += 256.
loop
mvzer ( TOP , 256. )



end
```

```
(*      Name : APDECL.MG

        This routine sets up the buffers required by the AP. *)

integer DBF ( 40 ) ; Data buffers coded as follows
integer N          ; Chaining counter
integer FCBCHN ( 500. ) ;Chaining space

record LINE_REC
        char    TOP ( 256. )
        dummy 1024.
        char    BOT ( 256. )
endrecord

(*      DBF ( I )        MEANING          VALUE

        DBF ( 0 )        DBMO             1
        DBF ( 1 )        DBME             2
        DBF ( 2 )        DBMO2            3
        DBF ( 3 )        DBMO3            4
        DBF ( 4 )        DBME2            5
        DBF ( 5 )        DBME3            6
        DBF ( 6 )        DBFTR            7
        DBF ( 7 )        Chaining Buffer  8
        DBF ( 8 )        Chaining Buffer  9



        DBF ( 9 )        OUT1             10 )
        DBF ( 10 )       OUT2             11 !
        DBF ( 11 )       OUT3             12 !
        DBF ( 12 )       OUT4             13 ! <--- OUTPUT DATA
        DBF ( 13 )       OUT5             14 !
        DBF ( 14 )       OUT6             15 !
        DBF ( 15 )       OUT7             16 )
        DBF ( 16 )       DBF1             17 )
        DBF ( 17 )       DBF2             18 !
        DBF ( 18 )       DBF3             19 ! <--- RAW DATA
        DBF ( 19 )       DBF4             20 !
        DBF ( 20 )       DBF5             21 )
        DBF ( 21 )       DBF6             22 )
        DBF ( 22 )       DBF7             23 !
        DBF ( 23 )       DBF8             24 ! <--- FILTERED DATA
        DBF ( 24 )       DBF9             25 !
        DBF ( 25 )       DBF10            26 )
        DBF ( 26 )       DBF11            27 )
        DBF ( 27 )       DBF12            28 !
        DBF ( 28 )       DBF13            29 ! <--- EDGE DATA
        DBF ( 29 )       DBF14            30 !
        DBF ( 30 )       DBF15            31 )
        DBF ( 31 )       DBFD             32
        DBF ( 32 )       HZER1            33
        DBF ( 33 )       HZER2            34
        DBF ( 34 )       ZER1             35
        DBF ( 35 )       ZER2             36
```

```
        DBF ( 36 )      VO1             37
        DBF ( 37 )      VE1             38
                                                *)

(*      This routine initilizes the data buffers
        in the AP with the first five lines of
        the raw image data                  *)

define READ_INIT
 local
        integer DBF1
 DBF1 := 17.            ; AP raw data buffers
 with M_EDGE
 ptr ( LINE_REC ) := WNDADR - 2304.
 WNDOFF off
 MAPW ( WNDB )
 ACHN ( ptr ( N ) , FCBCHN , 500. ) ; Start chaining
 iter 5
        AHIAB ( BOT , DBF ( 31 ) , 128. )       ; (LINE) --) DBF1
        AUPAK ( DBF1 , DBF ( 31 ) )
        ADNSN ( DBF1 )                  ; Determine Normalizing Coeff.
        ANRDB ( DBF1 )                  ; Normalize
        increment DBF1
        ptr ( LINE_REC ) += 256.
 loop
        ACEND                           ; End chaining



        AHIAB ( FCBCHN , 8. , N )
        AXCHN ( 8. )
        ARLDB ( 8. )
end

define ZERO_DBF
  AZRDB ( DBF ( 34 ) )
  AZRDB ( DBF ( 35 ) )
end
```

```
(* THE MAKE OF AP400 ARRAY PROCESSOR PRIMITIVES *)
APUSH RADIX
OCTAL
          MAKE     'AINIT  RSXFUNC 34
;         MAKE     'ARESET RSXFUNC 36
;         MAKE     'AABRT  RSXFUNC 40
          MAKE     'ACEND  RSXFUNC 42
          MAKE     'ACHN   RSXFUNC 44
;         MAKE     'ACTL   RSXFUNC 46
;         MAKE     'ASETW  RSXFUNC 50
;         MAKE     'AWAIT  RSXFUNC 52
;         MAKE     'AWFCB  RSXFUNC 54
          MAKE     'AICHN  RSXFUNC 56
          MAKE     'AEXIT  RSXFUNC 60
          MAKE     'AABHI  RSXFUNC 62
          MAKE     'AHIAB  RSXFUNC 64
          MAKE     'ADNSN  RSXFUNC 66
          MAKE     'ANRDB  RSXFUNC 70
          MAKE     'AADD   RSXFUNC 72
;         MAKE     'AMUL   RSXFUNC 74
;         MAKE     'AFTR2  RSXFUNC 76
;         MAKE     'AITR1  RSXFUNC 100
          MAKE     'AALDB  RSXFUNC 102
          MAKE     'AZRDB  RSXFUNC 104
;         MAKE     'ASQRT  RSXFUNC 106
;         MAKE     'AEXPE  RSXFUNC 110
          MAKE     'ARLDB  RSXFUNC 112



          MAKE     'ASUB   RSXFUNC 114
          MAKE     'AMULS  RSXFUNC 116
          MAKE     'ATFR1  RSXFUNC 120
          MAKE     'ACONV  RSXFUNC 122
          MAKE     'AEDGE  RSXFUNC 124
          MAKE     'AMOVE  RSXFUNC 126
          MAKE     'ADETS  RSXFUNC 130
          MAKE     'ADBDB  RSXFUNC 132
          MAKE     'APRUN  RSXFUNC 134
          MAKE     'ASCDB  RSXFUNC 136
;         MAKE     'AZERO  RSXFUNC 140
          MAKE     'AHZER  RSXFUNC 142
          MAKE     'AVZER  RSXFUNC 144
          MAKE     'AORDB  RSXFUNC 146
          MAKE     'AUPAK  RSXFUNC 150
          MAKE     'ASCRS  RSXFUNC 152
          MAKE     'AHORZ  RSXFUNC 154
          MAKE     'AVERZ  RSXFUNC 156
          MAKE     'AVCON  RSXFUNC 160
APOP
```

```
        .TITLE  LIBRARY ROUTINES FOR AP400 FUNTIONS
        .IDENT/01/
;
;FUNCTION:      ALLOWS THE AP400 FUNCTIONS TO BE CALLED FROM MAGIC
;
;AUTHOR:        CHETANA BUCH
;
;DATE:          AUG 9, 1982
;
;REVISIONS:
;
;       .SBTTL  VARIABLE STORAGE FOR CALL ARGUMENTS
        .PSECT  APDATA,D,RW
;
;
;INTERFACE BETWEEN MAGIC/L AND FORTRAN CALLING SEQUENCE
ARGLST: .BLKW   ^D10            ;ARGUMENT LIST ( POINTERS )
MGLST:  .BLKW   ^D10            ;ARGUMENT LIST ( VALUES )
SAVER5: .WORD   0               ;TEMPORARY STORAGE FOR R5
;
        .SBTTL  MAGIC/L CALLABLE AP400 ROUTINES
        .PSECT  APCODE
;
;
;AP RESOURCE MANAGEMENT ROUTINES
;
        .GLOBL  KINIT




;
AINIT.  JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #0,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KINIT
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
;       .GLOBL  XLOAD
;
;ALOAD::JSR     R0,SAVR         ;SAVE REGISTERS
;       MOV     #2,R0           ;# OF ARGUMENTS
;       JSR     PC,APMG4        ;SET INTERFACE
;       JSR     PC,KLOAD        ;
;       JSR     R0,RSTOR        ;RESTORE REGISTERS
;       RTS     PC
;
;
;
        .GLOBL  KRESET
;
ARESET::JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #0,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KRESET
```

```
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KABRT
;
AABRT:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #0,R0           ;# OF ARGUMENTS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KABRT        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KCEND
;
ACEND:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #0,R0           ;# OF ARGUMENTS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KCEND        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;




        .GLOBL  KCHN
;
ACHN::  JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG2        ;SET INTERFACE
        JSR     PC,KCHN         ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
.
        .GLOBL  KCTL
;
ACTL::  JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG2        ;SET INTERFACE
        JSR     PC,KCTL
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KSETW
,
ASETW:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG2        ;SET INTERFACE
```

```
        JSR     PC,KSETW        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KWAIT
;
AWAIT:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #0,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KWAIT        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KWFCB
;
AWFCB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG2        ;SET INTERFACE
        JSR     PC,KWFCB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;


;
        .GLOBL  KXCHN
;
AXCHN:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KXCHN        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KEXIT
;
AEXIT:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #0,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KEXIT
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
;AP DATA MEMORY DATA BUFFER MANAGEMENT ROUTINES
;
        .GLOBL  KALDB
;
```

```
AALDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #2,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KALDB        ;
        JSR     R0,RSTOR        ;RESTORE REEGISTERS
        RTS     PC
;
;
;
        .GLOBL  KRLDB
;
ARLDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KRLDB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KDBTS
;
ADBTS:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KDBTS        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS




        RTS     PC
;
;
;
;DATA TRANSFER ROUTINES
;
        .GLOBL  KABHI
;
AABHI:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #4,R0           ;# OF ARGS
        JSR     PC,APMG3        ;SET INTERFACE
        JSR     PC,KABHI        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KHIAB
;
AHIAB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG3        ;SET INTERFACE
        JSR     PC,KHIAB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
```

```
;
        .GLOBL  XMOVE
;
AMOVE:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KMOVE        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
.LOGICAL DATA MANIPULATION ROUTINES
;
        .GLOBL  KDNSN
;
ADNSN:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KDNSN        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
.
;
        .GLOBL  KNRDB
;




ANRDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KNRDB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KZRDB
;
AZRDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #1,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KZRDB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KDBDB
;
ADBDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #2,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KDBDB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
```

```
        RTS     PC
;
;
;
;COMPUTATION ROUTINES
;
        .GLOBL  KSCDB
;
ASCDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #2,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KSCDB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KTFR1
;
ATFR1:: JSR     R0,SAVR         ;SAVE REEGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KTFR1        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;



;
        .GLOBL  KCONV
;
ACONV:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KCONV        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KEDGE
;
AEDGE:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #5,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KEDGE        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KZERO
;
AZERO:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #5,R0           ;# OF ARGS
```

```
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KZERO        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KPRUN
;
APRUN:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #7,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KPRUN        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KHZER
;
AHZER:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;set interface
        JSR     PC,KHZER        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;



;
;       .GLOBL  KVZER
;
AVZER:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #6,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KVZER        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KORDB
;
AORDB:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #2,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KORDB        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KUPAK
;
AUPAK:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #2,R0           ;# OF ARGS
```

```
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KUPAK        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KSUB
;
ASUB::  JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KSUB         ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  XMULS
;
AMULS:: JSR     R0,SAVR         ;SAVE REEGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KMULS        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;



;
;
        .GLOBL  KADD
;
AADD::  JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KADD         ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  XMUL
;
AMUL::  JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KMUL         ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KSQRT
;
ASQRT:: JSR     R0,SAVR         ;SAVE REGISTERS
```

```
        MOV     #2,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KSQRT        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KEXPE
;
AEXPE:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #2,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KEXPE        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
        .GLOBL  KFTR2
;
AFTR2:: JSR     R0,SAVR         ;SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KFTR2        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS




        RTS     PC
;
;
;
        .GLOBL  KITR1
;
AITR1:: JSR     R0,SAVR         ,SAVE REGISTERS
        MOV     #3,R0           ;# OF ARGS
        JSR     PC,APMG1        ;SET INTERFACE
        JSR     PC,KITR1        ;
        JSR     R0,RSTOR        ;RESTORE REGISTERS
        RTS     PC
;
;
;
;SERVICE SUBROUTINES
;
;SAVE ALL REGSTERS
;
SAVR:   MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        JMP     (R0)
;
;
;RESTORE ALL REGISTERS
;
```

```
RSTOR:  TST     (SP)+
        MOV     (SP)+,R4
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     #-1,R1              ;REQ BY MAGIC
        MOV     SAVER5,R5           ;RESTORE R5
        RTS     R0
;
;
;
;MAGIC/L INTERFACE SETUP ROUTINES
;
;THIS ROUTINE IS CALLED FOR ALL FUNCTIONS WHOSE CALL STATEMENT
;ARGUMENTS ARE OF THE FOLLOWING TYPE:
;       ( NO ARGUMENT )
;       ( VAL )
;       ( VAL,VAL )
;       ( VAL,VAL,VAL )
;       ( VAL,VAL,VAL,VAL )
;       WHERE VAL IS ANY INTEGER VALUE.
;
APMG1:  MOV     R0,ARGLST           ;SET # OF ARGS
        TST     R0                  ;# OF ARGS
        BEQ     2$                  ;IF ZERO,TRANSFER CONTROL WITHOUT CHANGE
        MOV     R0,R1
        ASL     R1                  ;*2 FOR WORD ALLIGNMENT
        ADD     #ARGLST+2,R1        ;R1 POINTS TO ONEAFTER BOTTOM OF ARG PTR LIST



        MOV     #MGLST,R2           ;R2 POINTS TO THE ARGUMENT LIST ( MAGIC CALL )
1$:     MOV     R2,-(R1)            ;POINTER SET UP
        MOV     (R5)+,(R2)+         ;ARGUMENT SET UP
        DEC     R0
        BGT     1$
2$:     MOV     R5,SAVER5           ;SAVE R5
        MOV     #ARGLST,R5          ;FORTRAN CALL SET UP
        RTS     PC
;
;
;
;
;THIS ROUTINE IS CALLED BY FUNCTIONS WHOSE CALL STATEMENT
;ARGUMENTS ARE OF THE FOLLOWING TYPE:
;       ( ADDR )
;       ( ADDR,ADDR )
;       ( ADDR,ADDR,VAL )
;       WHERE ADDR IS A HOST MEMORY ADDRESS
;
APMG2:  MOV     R0,ARGLST           ;SET # OF ARGUMENTS
        MOV     R0,R1
        ADD     #-2,R1              ;TST # OF ARGS
        BLE     1$
        DEC     R0
        MOV     #MGLST,ARGLST+6     ;SET PTR TO VAL
        MOV     (R5)+,MGLST
1$:     DEC     R0
```

```
        BEQ     2$              ;IF 1 ARG ,NO CHANGE
        MOV     (R5)+,ARGLST+4  ;INTERCHANGE ARGUMENT ADDRESSES
                                ;MAGIC CALL R5 POINTS TO LAST ARG
2$:     MOV     (R5)+,ARGLST+2  ;FORTRAN CALL SETUP
        MOV     R5,SAVERS       ;SAVE R5
        MOV     #ARGLST,R5
        RTS     PC
;
;
.
;
;THIS ROUTINE IS CALLED BY FUNCTIONS WHOSE CALL STATEMENT
,ARGUMENTS ARE OF THE FOLLOWING TYPE:
;       ( ADDR,VAL,VAL )
;       ( ADDR,VAL,VAL,VAL )
;       WHERE ADDR IS A HOST MEMORY ADDRESS
;       AND VAL IS ANY INTEGER VALUE
;
APMG3:  MOV     R0,ARGLST       ;SET # OF ARGUMENTS
        MOV     R0,R1
        ASL     R1              ;*2 FOR WORD ALLIGNMENT
        ADD     #ARGLST+2,R1    ;R1 POINTS TO ONE AFTER THE BOTTOM OF
                                ;ARG PTR LIST ( FORTRAN CALL )
        MOV     #MGLST,R2       ;R2 POINTS TO ARG LIST (MAGIC CALL )
        DEC     R0
1$:     MOV     R2,-(R1)        ;ARG PTR SET UP
        MOV     (R5)+,(R2)+     ;ARG SET UP



        DEC     R0
        BGT     1$
        MOV     (R5)+,-(R1)
        MOV     R5,SAVERS       ;SAVE R5
        MOV     #ARGLST,R5      ;FORTRAN CALL SET UP
        RTS     PC
;
;
;
;
;THIS ROUTINE IS CALLED BY FUNCTIONS WHOSE CALL STATEMENT HAS
;ARGUMENTS OF THE FOLLOWING TYPE.
;       ( VAL,ADDR )
;       (VAL,ADDR,VAL )
;       WHERE ADDR IS A HOST MEMORY ADDRESS
;       AND VAL IS ANY INTEGER VALUE
;
APMG4:  MOV     R0,ARGLST       ;SET # OF ARGUMENTS
        ADD     #-2,R0          ;CHECK # OF ARGS
        BEQ     1$              ;FOR 2 ARGS
        MOV     #MGLST,ARGLST+6 ;SET PTR LIST
        MOV     (R5)+,MGLST
1$:     MOV     (R5)+,ARGLST+4  ;SET ADDRESS VALUE
        MOV     #MGLST+2,ARGLST+2
        MOV     (R5)+,MGLST+2
        MOV     R5,SAVERS       ;SAVE R5
        MOV     #ARGLST,R5      ;FORTRAN CALL SET UP
```

```
        RTS     PC
;
;
        .END







        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.
;-----------------------------------------------------------------------
;
;       PROGRAM:        HSTFNC. VECTOR ADD (REAL OR COMPLEX)
;
;       PART NUMBER:
;
;       VERSION DATE:   AUGUST 25, 1982
;
;       AUTHOR:         CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS FORTRAN-CALLABLE HOST FUNCTION CALLS UP AN AP-BASED
;                       AP FUNCTION IN ORDER TO PERFORM A "TIME DOMAIN CONVOLUTI
ON"
,       BETWEEN THE RESPECTIVE ELEMENTS OF TWO AP DATA MEMORY DATA BUFFERS. ONE

;       CONTAINS THE SIGNAL AND THE OTHER THE FILTER (MASK).
;
;-----------------------------------------------------------------------

        .TITLE  KCONV - HSTFNC: TIME CONVOLUTION

        .IDENT  /V01/           ;IDENTIFIER FOR THE OBJECT MODULE.
```

```
        .PAGE
;ESTABLISH ASSEMBLY AND LISTING CONVENTIONS:

        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .DSABL  GBL             ;FLAG NON-EXISTENT-SYMBOL REFERENCES AS ERRORS.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.

        .CSECT  KCONV           ;ESTABLISH A NAMED CSECT.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KCONV

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KEXFCB          ;AP MANAGER'S "FCB EXECUTION" SUBROUTINE.
        .GLOBL  KWAIT           ;AP MANAGER'S WAIT ROUTINE
        .GLOBL  MGRM67          ;AP MANAGER'S "FATAL ERROR #-67" EXIT ROUTINE.
        .GLOBL  COMCTL          ;AP MANAGER'S "FCB CONTROL WORD".

;AP FUNCTION ID'S REFERENCED:

CONV    =       802.    ;ID FOR "TIME CONVOLUTION".




;SYMBOL DEFINITIONS:

        ;NONE

;TERMINOLOGY:

;       FCB -   FUNCTION CONTROL BLOCK, READ BY THE AP EXECUTIVE FROM HOST
;               MEMORY.

        .PAGE
;)+HOST FUNCTION        "KCONV"

;THIS HOST FUNCTION CALLS UP A CORRESPONDING AP FUNCTION IN THE AP400.

;THIS HOST FUNCTION VERSION ASSUMES THAT SOURCE DATA ALREADY RESIDES IN TWO AP
;DATA MEMORY DATA BUFFERS, AND THAT THE RESULT DATA WILL BE PLACED IN ANOTHER
;AP DATA MEMORY DATA BUFFER.
;THE MAXIMUM MASK ( FILTER ) SIZE HANDLED BY THIS ROUTINE IS EIGHT POINTS. IF
;FILTER IS SMALLER, THE REMAINING BUFFER MUST CONTAIN ZEROS.
;THE TWO POINTS AT BOTH ENDS OF RESULT WILL CONTAIN ZEROS.
;THE CORRESPONDING "TIME CONVOLUTION" AP FUNCTION SHOULD BE
;REFERENCED FOR FURTHER INFORMATION.
```

```
;CALL FROM FORTRAN VIA:

;       SUBROUTINE CALL:        CALL    XCONV( DBIa, DBIb, DBIc )

;       OR INTEGER FUNCTION CALL, AS:   IERR = KCONV ( DBIa, DBIb, DBIc )

;WHERE:

;       DBIa =  ID OF AP DATA BUFFER TO HOLD RESULT DATA.
;               "DBIa" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF NEED NOT HAVE BEEN PREVIOUSLY ALLOCATED.
;               IF NOT ALREADY ALLOCATED, DBF WILL BE ALLOCATED; SIZE WILL EQUAL
;               THAT OF SOURCE DATA BUFFERS.
;               IF RESULT DBF WAS PREVIOUSLY ALLOCATED, IT MUST BE OF SIZE EQUAL
;               OR GREATER THAN SOURCE DATA BUFFERS.
;       DBIb =  ID OF AP DATA BUFFER HOLDING SIGNAL DATA SET.
;               "DBIb" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY.
;       DBIc =  ID OF AP DATA BUFFER HOLDING MASK ( FILTER ) DATA SET.
;               "DBIc" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY.
;               IF LESS THAN 8 PTS.,THE REMAINING BUFFER SHOULD BE ZEROED.
;RETURNS TO FORTRAN WITH:

;       ALL ARGUMENTS RETURNED AS RECEIVED.
;       FUNCTION EXECUTION "IN PROGRESS" OR "COMPLETE", DEPENDING UPON CURRENT


;               AP MANAGER "RETURN" STATUS.
;       IF CALLED AS A FORTRAN FUNCTION, THE VALUE RETURNED WILL BE AS SPECIFIED
;               FOR REGISTER "R0", RETURNED FROM AN ASSEMBLY-LANGUAGE CALL.

;       UPON ERROR, A STANDARD AP MANAGER ERROR EXIT WILL BE TAKEN.

;CALL FROM PDP-11 ASSEMBLY LANGUAGE VIA:

;       A FORTRAN-COMPATIBLE CALL SEQUENCE.

;RETURNS TO CALL+1:             (ALWAYS)

;       ALL CONDITIONS AS DESCRIBED FOR THE FORTRAN FUNCTION CALL FORM, ABOVE.
;       R0 =    STATUS VALUE.  (DEFINED BY AP MANAGER.)
;               "XCONV" DEFINES NO UNIQUE VALUES.
;       R1 =    UNDEFINED.
;       R2 =    UNDEFINED.
;       R5 =    UNDEFINED.

;UPON ERROR, WHEN CALLED FROM FORTRAN OR ASSEMBLY LANGUAGE:

;       IF A FATAL ERROR OCCURS DURING EXECUTION OF THIS HOST FUNCTION OR DURING
;       EXECUTION OF A ROUTINE WHICH IT (IN TURN) CALLS (SUCH AS THE AP MANAGER
;       OR AP DRIVER), THE AP MANAGER'S FATAL ERROR EXIT ROUTINE WILL BE CALLED.
;)-
```

```
        .PAGE
KCONV:

        CMPB    (R5), #3        ;CHECK FOR PROPER NUMBER OF ARGUMENTS.
        BNE     ERRORX          ;IF NOT CORRECT NUMBER, HANDLE AS A FATAL ERROR.

        TST     (R5)+           ;STEP POINTER AHEAD TO FIRST ARGUMENT ADDRESS.

        TST     FCBDON          ;TEST FOR COMPLETION OF A PREVIOUS OPERATION.
        BNE     1$              ;A ZERO "DONE" FLAG INDICATES PREVIOUS OPERATION
                                ;  STILL IN PROGRESS.
        JSR     PC,KWAIT                ;WAIT FOR THE AP TO FINISH PROCESSING

1$:     CLR     FCBDON          ;REINITIALIZE THE "DONE" FLAG.

        MOV     COMCTL, FCBCTL  ;RETRIEVE AP MANAGER'S COMMON CONTROL WORD IN
                                ;  ORDER TO UTILIZE CURRENTLY-SELECTED OPTIONS.
                                ;  PLACE IT IN FCB'S CONTROL WORD.

        MOV     @(R5)+, FCBARL  ;MOVE RESULT DATA BUFFER ID "A" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+4        ;MOVE SOURCE DATA BUFFER ID "B" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+8        ;MOVE SOURCE DATA BUFFER ID "C" INTO FCB
                                ;  ARGUMENT LIST.



                                ;  (INCREMENTING R5, ALTHOUGH UNNECESSARY, SAVES
                                ;  EXECUTION TIME AND ONE MEMORY WORD.)

        MOV     #MGRARG, R5     ;SET UP ADDRESS OF ARGUMENT LIST FOR CALL TO AP
                                ;  MANAGER.
        JMP     KEXFCB          ;CALL UP THE AP MANAGER TO PROCESS THE FCB.
                                ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR",
                                ;  FOLLOWED BY AN "RTS  PC".
                                ;  "KEXFCB" WILL RETURN ITS STATUS VALUE IN
                                ;  PDP-11 REGISTER R0 AS WELL AS IN LOCATION
                                ;  "STATUS".
MGRARG: BR      2$              ;BRANCH AROUND ARGUMENT LIST.  (THIS INSTRUCTION
                                ;  PROVIDES "NUMBER OF ARGUMENTS" COUNT FOR AP
                                ;  MANAGER; THE BRANCH IS NEVER ACTUALLY TAKEN.)
        .WORD   FCBBLK          ;ADDRESS OF FCB.
        .WORD   STATUS          ;ADDRESS FOR RETURNED STATUS.
2$:     ;THIS LABEL MARKS THE END OF THE ARGUMENT LIST.

ERRORX: JMP     MGRM67          ;TAKE AN AP MANAGER STANDARD FATAL ERROR EXIT.
                                ;  RETURN STATUS CODE -67 TO INDICATE "IMPROPER
                                ;  NUMBER OF ARGUMENTS IN PARAMETER LIST".

STATUS: .WORD   0               ;TEMPORARY STORAGE LOCATION FOR RETURNED AP
                                ;  MANAGER STATUS.
        .PAGE
;FUNCTION CONTROL BLOCK:
```

```
FCBBLK:

FCBID.   .WORD   CONV          ;ID OF THE AP FUNCTION.
FCBCTL:  .WORD   0             ;CONTROL WORD.
FCBDON:  .WORD   1             ;DONE FLAG.  INITIALIZED TO "DONE" STATE.
FCBLNK:  .WORD   0             ;(HIGH-ORDER.)  HOST MEMORY ADDRESS LINK TO NEXT
         .WORD   0             ;(LOW-ORDER.)   FCB IN HOST MEMORY.  (NONE.)

FCBPLT:  .WORD   1             ;FCB PARAMETER LIST TYPE.  (DATA BUFFER ID'S.)
FCBNRG:  .WORD   3             ;NUMBER OF ENTRIES IN ARGUMENT LIST.
FCBLEN:  .WORD   6             ;LENGTH OF ARGUMENT LIST IN HOST MEMORY WORDS.

FCBARL.  .WORD   0             ;RESULT DATA BUFFER ID "A" ARGUMENT.
         .WORD   0             ;  FIRST WORD = DBF ID; SECOND WORD = 0.

         .WORD   0             ;SIGNAL DATA BUFFER ID "B" ARGUMENT.
         .WORD   0             ;  FIRST WORD = DBF ID; SECOND WORD = 0.

         .WORD   0             ;FILTER DATA BUFFER ID "C" ARGUMENT.
         .WORD   0             ;  FIRST WORD = DBF ID; SECOND WORD = 0.

         .END




;--------------------------------------------------------------------------------
;
;       PROGRAM:        APFNC: TIME CONVOLUTION
;
;       PART NUMBER:
;
;       VERSION DATE:   AUGUST 25, 1982
;
;       AUTHORS:        CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS AP-BASED AP FUNCTION PERFORMS A TIME DOMAIN CONVOLU
TION
;       OF A SIGNAL WITH A FILTER OF MAXIMUM SIX POINTS
;
;       THIS AP FUNCTION IS NORMALLY CALLED UP BY THE AP EXECUTIVE, WHICH
;       RETRIEVES THIS AP FUNCTION'S ID NUMBER FROM A FUNCTION CONTROL BLOCK
;       READ FROM HOST MEMORY.
;
;--------------------------------------------------------------------------------


        TITLE   APFNC: TIME CONVOLUTION

        NAME    QCONV, 001      ;NAME AND VERSION FOR THE OBJECT MODULE.
```

```
        PAGE
        RADIX   H               ;DEFAULT TO HEXADECIMAL RADIX.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:          (IGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        ;NONE

;       GENERAL SYMBOLS

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:          (EGLOBL)

,       ENTRY POINTS:




        ;NONE

;       SUBROUTINES:

        EGLOBL  PLS1CE, ADDI1, FLSHAP, GETLZC, NRMCND

;       GENERAL SYMBOLS:

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE


;SYMBOL DEFINITIONS:

        ;NONE

;TERMINOLOGY:

        ;NONE
```

```
        PAGE
        PMORG           ;START OF RELOCATABLE CODE IN PROGRAM MEMORY.

;)+AP FUNCTION  "QCONV"

; This AP function performs the time convolution.

; Call with:      parameter list type     = 1,    number of arguments    = 3.
;                 parameter list length   = 6.

;       word 9  argument #1              = ID of result Data Buffer "A".
;       word 10 argument #1              = Ignored.

;       word 11 argument #2              = ID of source Data Buffer "B".
;       word 12 argument #2              = Ignored.

;       word 13 argument #3              = ID of source Data Buffer "C".
;       word 14 argument #3              = Ignored.

; Exits to AP Executive's "Fatal Abort" Service:

;       If an error is found by AP Service Subroutine 'PLSICE'.
;)-

        PAGE


;DEFINITION OF THE FUNCTION ID FOR THE AP EXECUTIVE FUNCTION TABLE:

        FUNC    %D802, QCONV    ;FUNCTION ID AND ENTRY POINT NAME.

PCLRAC: EQU     %D32            ;CLEAR ACC IN PIPE PAC ID.
PCONVS: EQU     %D82            ;CONVOLUTION (INITIAL) PAC ID.
PCONVT: EQU     %D83            ;CONVOLUTION (ITERATIVE) PAC ID.

QCONV:

        JSR     PLSICE          ;GO CHECK CORRECTNESS OF VALUES IN FCB,
                                ;  FIND SOURCE DATA BUFFERS,
                                ;  ALLOCATE RESULT DBF IF NECESSARY,
                                ;  SET UP ARGUMENTS FOR A FUNCTION ADDR  CALL.
                                ;  UPON ERROR, EXIT THROUGH AP EXECUTIVE'S
                                ;    FATAL ABORT ROUTINE.

        JSR     ADDI1           ;FORM AND STORE RESULT BEX AND NSN
                                ;
        SET     R2=R8+1         ;PTR TO FIRST RESULT DATA
        SETR    R3=0            ;
        STREGI  R3,R2           ;WRITE ZERO IN FIRST TWO PLACES
        STREG   R3,R2
        SET     R2=R2-1
        SET     R2=R2+R9        ;PTR TO LAST+1 RESULT DATA
        STREGD  R3,R2           ;WRITE ZERO IN LAST TWO PLACES
        STREGD  R3,R2           ;
```

```
        MOVE    REGSCL,R3          ;CLEAR SCL/LZC REGISTER

        SET     R10=R9-4           ;COUNT REGISTER
        SET     R11=R7-1           ;INITIALIZE SIGNAL DATA POINTER
        SET     R13=R8+2           ;INITIALIZE RESULT DATA POINTER

AGAIN:  SET     R12=R6+1           ;INITIALIZE FILTER DATA POINTER

        PIPE    PCLRAC,SCL0,LZCOFF        ;CLEAR PIPE ACCUMULATORS
        PAD     R3=R3
        PAD     ;NOT USED
        PAD     ;NOT USED
        PAD     ;NOT USED

        SETR    R2=2

        PIPE    PCONVS,SCL0,LZCOFF        ;CONV ( FOUR POINTS )
        PAD     R11=R11+R2,S1
        PAD     R11=R11+R2,S2
        PAD     R12=R12,S3
        PAD     R12=R12+R2,S4

        PIPE    PCONVT,SCL0,LZC2          ;REMAINING POINTS CONV
        PAD     R11=R11+R2,S1
        PAD     R13=R13+1,D2R
        PAD     R12=R12+R2,S3




        PAD     R12=R12+R2,S4

        SET     R11=R11-5          ;REINITIALIZE SIGNAL DATA PTR
        DBNZ    R10,AGAIN          ;REPEAT UNTILL ALL DATA DONE

        JSR     FLSHAP             ;FLUSH PIPELINE

        JSR     GETLZC             ;UPDATE NSN OF RESULT

        JMP     NRMCND             ;GO TO NORMALIZE THE RESULT DATA, IF FCB CONTROL
                                   ;  BIT INDICATES SUCH REQUIREMENT.
                                   ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR"
                                   ;  FOLLOWED BY AN "RTN".

        END
```

```
        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.
;---------------------------------------------------------------------------
;
;       PROGRAM:        HSTFNC: EDGE PRUNING
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 1, 1982
;
;       AUTHOR:         CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS FORTRAN-CALLABLE HOST FUNCTION CALLS UP AN AP-BASED
;                       AP FUNCTION IN ORDER TO PERFORM  "EDGE PRUNING"
;
;---------------------------------------------------------------------------

        .TITLE  KPRUN - HSTFNC: EDGE PRUNING

        .IDENT  /V01/           ;IDENTIFIER FOR THE OBJECT MODULE.

        .PAGE
;ESTABLISH ASSEMBLY AND LISTING CONVENTIONS:


        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .DSABL  GBL             ;FLAG NON-EXISTENT-SYMBOL REFERENCES AS ERRORS.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.

        .CSECT  KPRUN           ;ESTABLISH A NAMED CSECT.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KPRUN

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KEXFCB          ;AP MANAGER'S "FCB EXECUTION" SUBROUTINE.
        .GLOBL  KWAIT           ;AP MANAGER'S WAIT ROUTINE
        .GLOBL  MGRM67          ;AP MANAGER'S "FATAL ERROR #-67" EXIT ROUTINE.
        .GLOBL  COMCTL          ;AP MANAGER'S "FCB CONTROL WORD".

;AP FUNCTION ID'S REFERENCED:

PRUN    =       806.    ;ID FOR "VECTOR ADD (REAL OR COMPLEX)".

;SYMBOL DEFINITIONS:
```

```
        ;NONE

;TERMINOLOGY:

;       FCB -   FUNCTION CONTROL BLOCK, READ BY THE AP EXECUTIVE FROM HOST
;               MEMORY.

        .PAGE
;)+HOST FUNCTION         "KPRUN"

;THIS HOST FUNCTION CALLS UP A CORRESPONDING AP FUNCTION IN THE AP400

;THIS HOST FUNCTION VERSION ASSUMES THAT SOURCE DATA ALREADY RESIDES IN SEVEN AP
;DATA MEMORY DATA BUFFERS, AND THAT THE PRUNING WILL BE DONE ON DATA IN THE FOUR
TH
;AP DATA MEMORY DATA BUFFER.

;THE CORRESPONDING "EDGE PRUNING" AP FUNCTION SHOULD BE
,REFERENCED FOR FURTHER INFORMATION.

;CALL FROM FORTRAN VIA.

;       SUBROUTINE CALL:         CALL    KPRUN ( DBIa, DBIb,... DBIg )




;       OR INTEGER FUNCTION CALL, AS:   IERR = KADD ( DBIa, DBIb,... DBIg )

;WHERE:

;       DBIa,DBIb,DBIc...DBIg =
;
;                                  ID OF AP DATA BUFFERS WHICH HOLD RESULT EDGE
;               INFORMATION OF SEVEN CONSECUTIVE IMAGE LINES. DBId WILL HOLD
;               THE INFORMATION WHICH WILL BE THE FOCUS OF THIS PRUNER.
;               "DBIx" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED.
;
;       RETURNS TO FORTRAN WITH:

;       ALL ARGUMENTS RETURNED AS RECEIVED.
;       FUNCTION EXECUTION "IN PROGRESS" OR "COMPLETE", DEPENDING UPON CURRENT
,               AP MANAGER "RETURN" STATUS.
;       IF CALLED AS A FORTRAN FUNCTION, THE VALUE RETURNED WILL BE AS SPECIFIED
;               FOR REGISTER "R0", RETURNED FROM AN ASSEMBLY-LANGUAGE CALL.

.       UPON ERROR, A STANDARD AP MANAGER ERROR EXIT WILL BE TAKEN.

;CALL FROM PDP-11 ASSEMBLY LANGUAGE VIA:

;       A FORTRAN-COMPATIBLE CALL SEQUENCE.
```

```
;RETURNS TO CALL+1:              (ALWAYS)

;       ALL CONDITIONS AS DESCRIBED FOR THE FORTRAN FUNCTION CALL FORM, ABOVE.
;       R0 =    STATUS VALUE.  (DEFINED BY AP MANAGER.)
;               "KPRUN" DEFINES NO UNIQUE VALUES.
;       R1 =    UNDEFINED.
;       R2 =    UNDEFINED.
;       R5 =    UNDEFINED.


;UPON ERROR, WHEN CALLED FROM FORTRAN OR ASSEMBLY LANGUAGE:

;       IF A FATAL ERROR OCCURS DURING EXECUTION OF THIS HOST FUNCTION OR DURING
;       EXECUTION OF A ROUTINE WHICH IT (IN TURN) CALLS (SUCH AS THE AP MANAGER
;       OR AP DRIVER), THE AP MANAGER'S FATAL ERROR EXIT ROUTINE WILL BE CALLED.
;)-
        .PAGE
KPRUN:

        CMPB    (R5), #7        ;CHECK FOR PROPER NUMBER OF ARGUMENTS.
        BNE     ERRORX          ;IF NOT CORRECT NUMBER, HANDLE AS A FATAL ERROR.

        TST     (R5)+           ;STEP POINTER AHEAD TO FIRST ARGUMENT ADDRESS.

        TST     FCBDON          ;TEST FOR COMPLETION OF A PREVIOUS OPERATION.
        BNE     1$              ;A ZERO "DONE" FLAG INDICATES PREVIOUS OPERATION
                                ;  STILL IN PROGRESS.



        JSR     PC,KWAIT                ;WAIT FOR THE AP TO FINISH PROCESSING

1$:     CLR     FCBDON          ;REINITIALIZE THE "DONE" FLAG.

        MOV     COMCTL, FCBCTL  ;RETRIEVE AP MANAGER'S COMMON CONTROL WORD IN
                                ;  ORDER TO UTILIZE CURRENTLY-SELECTED OPTIONS.
                                ;  PLACE IT IN FCB'S CONTROL WORD.

        MOV     @(R5)+, FCBARL  ;MOVE RESULT DATA BUFFER ID "A" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+4        ;MOVE SOURCE DATA BUFFER ID "B" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+8.       ;MOVE SOURCE DATA BUFFER ID "C" INTO FCB
                                ;  ARGUMENT LIST.
        MOV     @(R5)+, FCBARL+12.      ;MOVE SOURCE DATA BUFFER ID "D" INTO FCB
                                ;  ARGUMENT LIST.
        MOV     @(R5)+, FCBARL+16.      ;MOVE SOURCE DATA BUFFER ID "E" INTO FCB
                                ;  ARGUMENT LIST.
        MOV     @(R5)+, FCBARL+20.      ;MOVE SOURCE DATA BUFFER ID "F" INTO FCB
                                ;  ARGUMENT LIST.
        MOV     @(R5)+, FCBARL+24.      ;MOVE SOURCE DATA BUFFER ID "G" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  (INCREMENTING R5, ALTHOUGH UNNECESSARY, SAVES
                                ;  EXECUTION TIME AND ONE MEMORY WORD.)
```

```
        MOV     #MGRARG, R5     ;SET UP ADDRESS OF ARGUMENT LIST FOR CALL TO AP
                                ;  MANAGER.
        JMP     KEXFCB          ;CALL UP THE AP MANAGER TO PROCESS THE FCB.
                                ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR",
                                ;  FOLLOWED BY AN "RTS  PC".
                                ;  "KEXFCB" WILL RETURN ITS STATUS VALUE IN
                                ;  PDP-11 REGISTER R0 AS WELL AS IN LOCATION
                                ;  "STATUS".
MGRARG: BR      2$              ;BRANCH AROUND ARGUMENT LIST.  (THIS INSTRUCTION
                                ;  PROVIDES "NUMBER OF ARGUMENTS" COUNT FOR AP
                                ;  MANAGER; THE BRANCH IS NEVER ACTUALLY TAKEN.)
        .WORD   FCBBLK          ;ADDRESS OF FCB.
        .WORD   STATUS          ;ADDRESS FOR RETURNED STATUS.
2$:     ;THIS LABEL MARKS THE END OF THE ARGUMENT LIST.

ERRORX: JMP     MGRM67          ;TAKE AN AP MANAGER STANDARD FATAL ERROR EXIT.
                                ;  RETURN STATUS CODE -67 TO INDICATE "IMPROPER
                                ;  NUMBER OF ARGUMENTS IN PARAMETER LIST".

STATUS: .WORD   0               ;TEMPORARY STORAGE LOCATION FOR RETURNED AP
                                ;  MANAGER STATUS.
        .PAGE
;FUNCTION CONTROL BLOCK:

FCBBLK:

FCBID:  .WORD   PRUN            ;ID OF THE AP FUNCTION.




FCBCTL: .WORD   0               ;CONTROL WORD.
FCBDON: .WORD   1               ;DONE FLAG.  INITIALIZED TO "DONE" STATE.
FCBLNK: .WORD   0               ;(HIGH-ORDER.)  HOST MEMORY ADDRESS LINK TO NEXT
        .WORD   0               ;(LOW-ORDER.)   FCB IN HOST MEMORY.  (NONE.)

FCBPLT: .WORD   1               ;FCB PARAMETER LIST TYPE.  (DATA BUFFER ID'S.)
FCBNRG: .WORD   7               ;NUMBER OF ENTRIES IN ARGUMENT LIST.
FCBLEN: .WORD   14.             ;LENGTH OF ARGUMENT LIST IN HOST MEMORY WORDS.

FCBARL: .WORD   0               ;RESULT DATA BUFFER ID "A" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "B" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "C" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "D" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "E" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "F" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.
```

```
        .WORD   0               ;SOURCE DATA BUFFER ID "C" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .END
```

```
;----------------------------------------------------------------------------
;
;       PROGRAM:        APFNC: EDGE PRUNING
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 1, 1982
;
;       AUTHORS:        CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS AP-BASED AP FUNCTION PERFORMES EDGE PRUNING.
;                       SEVEN DATA BUFFERS CONTAINING EDGE INFORMATION OF SEVEN
;       CONSECUTIVE IMAGE DATA LINES ARE REQUIRED.
;
;       THIS AP FUNCTION IS NORMALLY CALLED UP BY THE AP EXECUTIVE, WHICH
;       RETRIEVES THIS AP FUNCTION'S ID NUMBER FROM A FUNCTION CONTROL BLOCK
;       READ FROM HOST MEMORY.
;
;----------------------------------------------------------------------------

        TITLE   APFNC: EDGE PRUNING

        NAME    OPRUN, 001      ;NAME AND VERSION FOR THE OBJECT MODULE.
```

```
        PAGE
        RADIX   H                 ;DEFAULT TO HEXADECIMAL RADIX.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:          (IGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        ;NONE

;       GENERAL SYMBOLS

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:          (EGLOBL)

;       ENTRY POINTS:




        ;NONE

;       SUBROUTINES:

        EGLOBL  PLCHK1, PLDBF, FTLABT, NRMCND

;       GENERAL SYMBOLS:

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE

;SYMBOL DEFINITIONS:

        ;NONE

;TERMINOLOGY:

        ;NONE
```

```
        PAGE
        PMORG            ;START OF RELOCATABLE CODE IN PROGRAM MEMORY.

;)+AP FUNCTION  "QPRUN"

; This AP Function scans the data in DBId to find odd zero crossings ( results
; from QEDGE ) and deletes any even zero crossings in the vicinity of three pixe
ls.

; Call with:     parameter list type     = 1,    number of arguments    = 7,
;                parameter list length   = 14.

;       word 9  argument #1              = ID of result Data Buffer "A".
;       word 10 argument #1              = Ignored.

;       word 11 argument #2              = ID of source Data Buffer "B".
;       word 12 argument #2              = Ignored.

;       word 13 argument #3              = ID of source Data Buffer "C".
;       word 14 argument #3              = Ignored.

;       word 15 argument #4              = ID of source Data Buffer "D".
;       word 16 argument #4              = Ignored.

;       word 17 argument #5              = ID of source Data Buffer "E".
;       word 18 argument #5              = Ignored.

;       word 19 argument #6              = ID of source Data Buffer "F".
;       word 20 argument #6              = Ignored.

;       word 21 argument #7              = ID of source Data Buffer "G".
;       word 22 argument #7              = Ignored.

; Exits to AP Executive's "Fatal Abort" Service:

;       If an error is found by AP Service Subroutine 'PLCHK1'or 'PLDBF'.
;)-

        PAGE
;DEFINITION OF THE FUNCTION ID FOR THE AP EXECUTIVE FUNCTION TABLE:

        FUNC    %D806, QPRUN    ;FUNCTION ID AND ENTRY POINT NAME.

QPRUN:
        SETR    R1=1            ;SET UP FOR PLCHK1 CALL
        SETR    R2=7
        SETR    R3=%D14
        JSR     PLCHK1          ;GO CHECK CORRECTNESS OF VALUES IN FCB,
                                ;  FIND SOURCE DATA BUFFERS,
                                ;  ALLOCATE RESULT DBF IF NECESSARY,
                                ;  SET UP ARGUMENTS FOR A FUNCTION ADDR. CALL.
```

```
                                    ;  UPON ERROR, EXIT THROUGH AP EXECUTIVE'S
        JMP     FTLABT              ;     FATAL ABORT ROUTINE.

        SETR    R15=7               ;ARG COUNT
FETCH:  JSR     PLDBF               ;FETCH DBF ADDR
        JMP     FTLABT              ;
        SET     R1=R1+1             ;POINT TO FIRST DATA WORD
        PUSH    R1                  ;SAVE ON STACK
        DBNZ    R15,FETCH

        POP     R14                 ;R14-->DBIg
        POP     R13                 ;R13-->DBIf
        POP     R12                 ;R12-->DBIe
        POP     R11                 ;R11-->DBId ... MAIN LINE...

        SETR    R3=STORE            ;SAVE OTHER ADDR IN STORE
        SETR    R4=3
SAVE:   POP     R1
        STREGI  R1,R3
        DBNZ    R4,SAVE

        SETR    R1=0                ;COUNT POINTER

NEXT:   LDREGI  R3,R11              ;GET DATA INTO R3
        SET     R4=R3               ;COPY IN R4
        SET     R4=R4'AND'%H3D      ;CHECK FOR CORNER
        SKIPNE  R4'XOR'%H3D




        JMP     CORNER
        SET     R4=R3
        SET     R4=R4'AND'%H39
        SKIPNE  R4'XOR'%H39         ;ODD HORZ CODE
        JMP     HORODD
        SET     R4=R3
        SET     R4=R4'AND'%H35
        SKIPNE  R4'XOR'%H35         ;ODD VERT CODE
        JMP     VERODD

DONE:   SET     R1=R1+1             ;UPDATE POINTER
        DBNZ    R2,NEXT

;       JMP     NRMCND              ;RETURN TO AP EXEC.

        RTN

CORNER: SETR    R15=-1              ;FLAG FOR CORNER PIXEL

HORODD: SETR    R4=-1               ;DIRECTION FLAG
        SETR    R5=3                ;COUNT
        SET     R6=R11              ;POINTER IN FORWARD DIRECTION

CHK:    LDREGI  R7,R6               ;FETCH DATA
CHK1:   SET     R8=R7               ;COPY IT
        SKIPNE  R8'XOR'%H3A         ;EVEN HORZ POSITIVE STRONG CODE
```

```
        JMP      ZEROH
        SET      R8=R7
        SKIPNE   R8'XOR'%H3E      ;EVEN CORNER STRONG CODE
        JMP      ZEROH

        SKIPLT   R4=R4            ;CHECK DIRECTION
        JMP      OPP
        DBNZ     R5,CHK           ;REPEAT

REV:    SETR     R4=0             ;REVERSE DIRECTION FLAG
        SETR     R5=3             ;COUNT
        SET      R6=R11-1
CHK2:   LDREGD   R7,R6
        JMP      CHK1

OPP:    DBNZ     R5,CHK2
        SKIPLT   R15=R15
        JMP      DONE
        SETR     R15=0
        JMP      VERODD


ZEROH.  SETR     R7=0             ;KILL THE EVEN CROSSING PRESENT
        SKIPLT   R4=R4
        JMP      LEFT
        SET      R6=R6-1          ;CORRECT POINTER
        STREG    R7,R6




        JMP      REV              ;CHECK IN OTHER DIRECTION
LEFT:   STREG    R7,R6
        SKIPLT   R15=R15
        JMP      DONE
        SETR     R15=0

VERODD: SETR     R4=-1            ;DIRECTION FLAG
        SETR     R5=3             ;COUNT
        SET      R6=R12
REPEAT: SET      R6=R6+R1         ;PTR TO CORRES WORD IN ADJECENT LINE
CHK3:   LDREG    R7,R6            ;FETCH DATA
        SET      R8=R7            ;COPY IT
        SKIPNE   R8'XOR'%H36      ;EVEN VERT POSITIVE STRONG CODE
        JMP      ZEROV
        SET      R8=R7
        SKIPNE   R8'XOR'%H3E      ;EVEN CORNER STRONG CODE
        JMP      ZEROV
        DBNZ     R5,TEST
        SKIPLT   R4=R4            ;CHECK DIRECTION
        JMP      DONE
REV1:   SETR     R4=0             ;FLAG FOR REVERSE DIRECTION
        SETR     R5=3             ;COUNT
        SETR     R9=STORE         ;FETCH OTHER ADDR
        LDREGI   R6,R9
        JMP      REPEAT
TEST:   SET      R7=R5
        SKIPEQ   R7'XOR'%H2
```

```
        JMP     THIRD
        SKIPLT  R4=R4
        JMP     REV2
        SET     R6=R13
        JMP     REPEAT          ;PTR TO NEXT LINE
THIRD:  SKIPLT  R4=R4
        JMP     REV3
        SET     R6=R14
        JMP     REPEAT          ;PTR TO THIRD LINE

REV2:   LDREGI  R6,R9
        JMP     REPEAT
REV3:   LDREGI  R6,R9
        JMP     REPEAT

ZEROV:  SETR    R7=0            ;KILL THE EVEN ZERO CROSSING
        STREG   R7,R6
        SKIPLT  R4=R4
        JMP     DONE
        JMP     REV1            ;CHECK IN REVERSE DIRECTION

STORE:  DS      0               ;STORE FOR BUFFER ADDRESSES
        DS      0
        DS      0

        END




        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.
;------------------------------------------------------------------------
;
;       PROGRAM:        HSTFNC: EDGE DETECTION FOR RAW IMAGE
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 13, 1982
;
;       AUTHOR:         CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS FORTRAN-CALLABLE HOST FUNCTION CALLS UP AN AP-BASED
;                       AP FUNCTION IN ORDER TO PERFORM  "EDGE DETECTION"
;       OPERATION BETWEEN THE RESPECTIVE ELEMENTS OF TWO AP DATA MEMORY DATA BUF
FERS
;       WHICH CONTAIN THE VARIOUS CONVOLUTION RESULTS OF THE LINE IMAGE WITH A M
ASK.
;
;------------------------------------------------------------------------

        .TITLE  KHORZ - HSTFNC: EDGE DETECTION

        .IDENT  /V01/           ;IDENTIFIER FOR THE OBJECT MODULE.
```

```
        .PAGE
;ESTABLISH ASSEMBLY AND LISTING CONVENTIONS:

        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .DSABL  GBL             ;FLAG NON-EXISTENT-SYMBOL REFERENCES AS ERRORS.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.

        .CSECT  KHORZ           ;ESTABLISH A NAMED CSECT.


;INTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KHORZ


;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KEXFCB          ;AP MANAGER'S "FCB EXECUTION" SUBROUTINE.
        .GLOBL  KWAIT           ;AP MANAGER'S WAIT ROUTINE
        .GLOBL  MGRM67          ;AP MANAGER'S "FATAL ERROR #-67" EXIT ROUTINE.
        .GLOBL  COMCTL          ;AP MANAGER'S "FCB CONTROL WORD".


;AP FUNCTION ID'S REFERENCED:

HORZ=   ^D820.          ;ID FOR "EDGE DETECTION".


;SYMBOL DEFINITIONS:

        ;NONE


;TERMINOLOGY:

;       FCB -   FUNCTION CONTROL BLOCK, READ BY THE AP EXECUTIVE FROM HOST
;               MEMORY.


        .PAGE
;)+HOST FUNCTION        "KHORZ"

;THIS HOST FUNCTION CALLS UP A CORRESPONDING AP FUNCTION IN THE AP400.

;THIS HOST FUNCTION VERSION ASSUMES THAT SOURCE DATA ALREADY RESIDES IN TWO AP
;DATA MEMORY DATA BUFFERS, AND THAT THE RESULT DATA WILL BE PLACED IN ANOTHER
;AP DATA MEMORY DATA BUFFER.

;THE CORRESPONDING "EDGE DETECTION" AP FUNCTION SHOULD BE REFERENCED FOR FURTHER
;INFORMATION.


;CALL FROM FORTRAN VIA:
```

```
;       SUBROUTINE CALL:        CALL    XHORZ ( DBIa, DBIb, DBIc )

;       OR INTEGER FUNCTION CALL, AS:   IERR = XHORZ ( DBIa, DBIb, DBIc )

;WHERE:

;       DBIa =  ID OF AP DATA BUFFER TO HOLD RESULT DATA.
;               "DBIa" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF NEED NOT HAVE BEEN PREVIOUSLY ALLOCATED.
;               IF NOT ALREADY ALLOCATED, DBF WILL BE ALLOCATED; SIZE WILL EQUAL
;               THAT OF SOURCE DATA BUFFERS.
;               IF RESULT DBF WAS PREVIOUSLY ALLOCATED, IT MUST BE OF SIZE EQUAL
;               OR GREATER THAN SOURCE DATA BUFFERS.
;       DBIb =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET( ODD HORZ. CONV RES
ULTS ).
;               "DBIb" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY
;               DATA BUFFERS DBIb,DBIc,DBId,DBIe MUST BE OF EQUAL LENGTH.
;       DBIc =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET(EVEN HORZ. CONV RES
ULTS).
;               "DBIc" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY.

;RETURNS TO FORTRAN WITH:

;       ALL ARGUMENTS RETURNED AS RECEIVED.


;       FUNCTION EXECUTION "IN PROGRESS" OR "COMPLETE", DEPENDING UPON CURRENT
;               AP MANAGER "RETURN" STATUS.
;       IF CALLED AS A FORTRAN FUNCTION, THE VALUE RETURNED WILL BE AS SPECIFIED
;               FOR REGISTER "R0", RETURNED FROM AN ASSEMBLY-LANGUAGE CALL.

;       UPON ERROR, A STANDARD AP MANAGER ERROR EXIT WILL BE TAKEN.


;CALL FROM PDP-11 ASSEMBLY LANGUAGE VIA:

;       A FORTRAN-COMPATIBLE CALL SEQUENCE.

;RETURNS TO CALL+1:             (ALWAYS)

;       ALL CONDITIONS AS DESCRIBED FOR THE FORTRAN FUNCTION CALL FORM, ABOVE.
;       R0 =    STATUS VALUE.  (DEFINED BY AP MANAGER.)
;               "XHORZ" DEFINES NO UNIQUE VALUES.
;       R1 =    UNDEFINED.
;       R2 =    UNDEFINED.
;       R5 =    UNDEFINED.


;UPON ERROR, WHEN CALLED FROM FORTRAN OR ASSEMBLY LANGUAGE:

;       IF A FATAL ERROR OCCURS DURING EXECUTION OF THIS HOST FUNCTION OR DURING
;       EXECUTION OF A ROUTINE WHICH IT (IN TURN) CALLS (SUCH AS THE AP MANAGER
;       OR AP DRIVER), THE AP MANAGER'S FATAL ERROR EXIT ROUTINE WILL BE CALLED.
```

```
.;)-
        .PAGE
KHORZ:

        CMPB    (R5), #3        ;CHECK FOR PROPER NUMBER OF ARGUMENTS.
        BNE     ERRORX          ;IF NOT CORRECT NUMBER, HANDLE AS A FATAL ERROR.

        TST     (R5)+           ;STEP POINTER AHEAD TO FIRST ARGUMENT ADDRESS.

        TST     FCBDON          ;TEST FOR COMPLETION OF A PREVIOUS OPERATION.
        BNE     1$              ;A ZERO "DONE" FLAG INDICATES PREVIOUS OPERATION
                                ;  STILL IN PROGRESS.
        JSR     PC,KWAIT              ;WAIT FOR THE AP TO FINISH PROCESSING

1$:     CLR     FCBDON          ;REINITIALIZE THE "DONE" FLAG.

        MOV     COMCTL, FCBCTL  ;RETRIEVE AP MANAGER'S COMMON CONTROL WORD IN
                                ;  ORDER TO UTILIZE CURRENTLY-SELECTED OPTIONS.
                                ;  PLACE IT IN FCB'S CONTROL WORD.

        MOV     @(R5)+, FCBARL  ;MOVE RESULT DATA BUFFER ID "A" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+4       ;MOVE SOURCE DATA BUFFER ID "B" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+^D8     ;MOVE SOURCE DATA BUFFER ID "C" INTO FCB




                                ;  ARGUMENT LIST.

                                ;  (INCREMENTING R5, ALTHOUGH UNNECESSARY, SAVES
                                ;  EXECUTION TIME AND ONE MEMORY WORD.)
        MOV     #MGRARG, R5     ;SET UP ADDRESS OF ARGUMENT LIST FOR CALL TO AP
                                ;  MANAGER.
        JMP     KEXFCB          ;CALL UP THE AP MANAGER TO PROCESS THE FCB.
                                ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR",
                                ;  FOLLOWED BY AN "RTS  PC".
                                ;  "KEXFCB" WILL RETURN ITS STATUS VALUE IN
                                ;  PDP-11 REGISTER R0 AS WELL AS IN LOCATION
                                ;  "STATUS".
MGRARG: BR      2$              ;BRANCH AROUND ARGUMENT LIST.  (THIS INSTRUCTION
                                ;  PROVIDES "NUMBER OF ARGUMENTS" COUNT FOR AP
                                ;  MANAGER; THE BRANCH IS NEVER ACTUALLY TAKEN.)
        .WORD   FCBBLK          ;ADDRESS OF FCB.
        .WORD   STATUS          ;ADDRESS FOR RETURNED STATUS.
2$:     ;THIS LABEL MARKS THE END OF THE ARGUMENT LIST.

ERRORX: JMP     MGRM67          ;TAKE AN AP MANAGER STANDARD FATAL ERROR EXIT.
                                ;  RETURN STATUS CODE -67 TO INDICATE "IMPROPER
                                ;  NUMBER OF ARGUMENTS IN PARAMETER LIST".

STATUS: .WORD   0               ;TEMPORARY STORAGE LOCATION FOR RETURNED AP
                                ;  MANAGER STATUS.

        .PAGE
;FUNCTION CONTROL BLOCK:
```

```
FCBBLK:

FCBID:  .WORD   HORZ            ;ID OF THE AP FUNCTION.
FCBCTL: .WORD   0               ;CONTROL WORD.
FCBDON: .WORD   1               ;DONE FLAG.  INITIALIZED TO "DONE" STATE.
FCBLNK: .WORD   0               ;(HIGH-ORDER.)  HOST MEMORY ADDRESS LINK TO NEXT
        .WORD   0               ;(LOW-ORDER.)   FCB IN HOST MEMORY.  (NONE.)

FCBPLT: .WORD   1               ;FCB PARAMETER LIST TYPE.  (DATA BUFFER ID'S.)
FCBNRG: .WORD   3               ;NUMBER OF ENTRIES IN ARGUMENT LIST.
FCBLEN: .WORD   6               ;LENGTH OF ARGUMENT LIST IN HOST MEMORY WORDS.

FCBARL: .WORD   0               ;RESULT DATA BUFFER ID "A" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "B" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "C" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .END




;----------------------------------------------------------------------------
;
;       PROGRAM:        APFNC: EDGE DETECTION
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 13, 1982
;
;       AUTHORS:        CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS AP-BASED AP FUNCTION PERFORMS AN EDGE DETECTION BY
;                       BASICALLY DETECTING A ZERO CROSSING IN THE CONVOLVED RES
ULTS
;       OF THE LINE OF RAW IMAGE DATA. ODD AND EVEN MASKS ARE USED ON HORIZONTAL
L
;       IMAGE DATA .
;       THE RESULT BUFFER CONTAINS A CODED WORD FOR EACH PIXEL.
;
;       THIS AP FUNCTION IS NORMALLY CALLED UP BY THE AP EXECUTIVE, WHICH
;       RETRIEVES THIS AP FUNCTION'S ID NUMBER FROM A FUNCTION CONTROL BLOCK
;       READ FROM HOST MEMORY.
;
;----------------------------------------------------------------------------
```

```
        TITLE   APFNC: EDGE DETECTION
        NAME    QHORZ.001        ;NAME AND VERSION FOR THE OBJECT MODULE.

        PAGE
        RADIX   H                ;DEFAULT TO HEXADECIMAL RADIX.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:          (IGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        ;NONE

;       GENERAL SYMBOLS

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE



;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:          (EGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        EGLOBL  PLCHK1, FTLABT, PLDBF, NRMCND

;       GENERAL SYMBOLS:

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE

;SYMBOL DEFINITIONS:

        ;NONE

;TERMINOLOGY:
```

```
        ;NONE

        PAGE
        PMORG           ;START OF RELOCATABLE CODE IN PROGRAM MEMORY.

;}+AP FUNCTION  "QHORZ"

; This AP Function performs an edge detection. This is actually a zero crossing
; detection scheme.

; Call with:    parameter list type      = 1,    number of arguments     = 3.
;               parameter list length    = 6.

;       word 9  argument #1              = ID of result Data Buffer "A".
;       word 10 argument #1              = Ignored.

;       word 11 argument #2              = ID of source Data Buffer "B".
;       word 12 argument #2              = Ignored.

;       word 13 argument #3              = ID of source Data Buffer "C".
;       word 14 argument #3              = Ignored.

; Exits to AP Executive's "Fatal Abort" Service:




;       If an error is found by AP Service Subroutine 'PLDBF' or 'PLCHK1'.
;}-

        PAGE
;DEFINITION OF THE FUNCTION ID FOR THE AP-EXECUTIVE FUNCTION TABLE:

        FUNC    %D820, QHORZ    ;FUNCTION ID AND ENTRY POINT NAME.

QHORZ:
                                ;SET UP FOR CALL TO PLCHK1
        SETR    R1=1            ;PARAMETER DESCRIPTOR TYPE
        SETR    R2=3            ;# OF ARGUMENTS
        SETR    R3=6            ;# OF WORDS IN ARG LIST
        JSR     PLCHK1          ;GO CHECK CORRECTNESS OF VALUES IN FCB.

        JMP     FTLABT          ;RETURNS HERE IF ERROR
                                ;IF OK, RETURNS HERE
        JSR     PLDBF           ;  FIND SOURCE DATA BUFFERS,
                                ;  ALLOCATE RESULT DBF IF NECESSARY,
                                ;  SET UP ARGUMENTS FOR A FUNCTION ADDR. CALL.
                                ;  UPON ERROR, EXIT THROUGH AP EXECUTIVE'S
                                ;     FATAL ABORT ROUTINE.
        JMP     FTLABT
```

```
        SET     R15=R1          ; R15--)RESULT BUFFER ADDRESS.

        JSR     PLDBF
        JMP     FTLABT
        SET     R13=R1+1        ; R13--)HORZ. ODD CONV ( Ho ) BUFFER ADDRESS.

        JSR     PLDBF
        JMP     FTLABT
        SET     R14=R1+1        ; R14--)HORZ. EVEN CONV ( He ) BUFFER ADDRESS.

        SETR    R1=%H0F
        STREG   R1,R15          ;SET BEX OF RESULT BUFFER
        SETR    R1=0
        STREGI  R1,R15,LO               ;SET NSN OF RESULT

        SET     R12=R2          ; R12--)BUFFER LENGTH.

START:  LDREGI  R3,R13          ;GET FIRST Ho/Vo VALUE
        LDREGI  R4,R14          ;GET FIRST He/Ve VALUE

        STREGI  R1,R15          ;OUTPUT ZERO FOR FIRST VALUE
        SET     R12=R12-1       ;DECR COUNT

CHKO:   SKIPLT  R5=R3           ;SIGN CHECK FOR ODD VALUES
        JMP     POSODD
        LDREGI  R3,R13




CHKE:   SKIPGE  R5=R4           ;SIGN CHECK FOR EVEN VALUES
        JMP     NEGEVN
        JMP     POSEVN
                                ;HERE IF VALUE IS ODD AND POSITIVE
POSODD: LDREGI  R3,R13          ;FETCH NEXT VALUE AND COMPARE WITH PREVIOUS
        SKIPLT  R3=R3
        JMP     CHKE            ;CHECK EVEN IF NO SIGN CHANGE
        SET     R11=R15
        SKIPGE  R5=R5+R3
        SET     R11=R11-1
        JMP     SELODD          ;ELSE SELECT APPROPRIATE CODE FOR OUTPUT

                                ;HERE IF VALUE IS EVEN AND POSITIVE
POSEVN: LDREGI  R4,R14          ;FETCH NEXT VALUE AND COMPARE WITH PREVIOUS
        SKIPLT  R4=R4
        JMP     OUTZ            ;OUTPUT ZERO, SINCE NO SIGN CHANCE
        SET     R2=R13-1
        SET     R11=R15
        SKIPGE  R5=R5+R4
        SET     R2=R2-1
        SET     R11=R11-1
        JMP     OUTEVN          ;ELSE SELECT APPROPRIATE CODE TO OUTPUT

                                ;HERE IF VALUE IS EVEN AND NEGATIVE
NEGEVN: LDREGI  R4,R14          ;FETCH NEXT VALUE AND COMPARE WITH PREVIOUS
        SKIPGE  R4=R4
        JMP     OUTZ            ;OUTPUT ZERO, SINCE NO SIGN CHANCE
```

```
         SET      R11=R15
         SET      R2=R13-1
         SKIPLT   R5=R5+R4
         SET      R11=R11-1
         SET      R2=R2-1
         JMP      OUTEVN            ;ELSE SELECT APPROPRIATE CODE TO OUTPUT

SELODD:  SET      R2=R14-1          ;POINTER TO EVEN DATA
                                    ;GETS THE RIGHT STRENGTH
         LDREGI   R5,R2             ;OBTAIN THE THREE CONSECUTIVE DATA VALUES
         LDREGI   R6,R2             ;IN R5, R6, R7.
         SKIPGE   R5=R5
         SETR     R5=0              ;ZERO IF NEGATIVE
         SKIPGE   R6=R6
         SETR     R6=0
         SET      R7=R6
         SKIPGE   R6=R6-R5          ;COMPARE TWO STRENGTHS
         SET      R7=R5             ;R5 IS LARGER

OUTODD:                             ;POSITIVE ODD ZERO CROSSINGS
         SETR     R5=%H3B           ;CODE FOR HORZ POS WEAK ODD PIXEL
         SETR     R8=%D600          ;NOISE THRESHOLD FOR ODD
         SKIPGE   R7=R7-R8          ;CHECK THRESHOLD
         JMP      CHKE              ;TOO LOW , NOT VALID SO CHECK EVEN
         LDREGI   R4,R14            ;INCR EVEN PTR SINCE NO EVEN CHECK DONE
         STREG    R5,R11            ;STRONG CODE IN OUT BUFFER
         JMP      NEXT




OUTEVN:  SETR     R6=%H3A           ;CODE FOR HORZ NEG EVEN STRONG PIXEL
         LDREG    R5,R2             ;FETCH STRENGTH

         SETR     R8=%D1000         ;NOISE THRESHOLD FOR EVEN
         SKIPGE   R5=R5-R8          ;CHECK THRESHOLD
         JMP      OUTZ              ;TOO LOW, SO WRITE OUT ZERO
         STREG    R6,R11            ;STRONG CODE
         JMP      NEXT

OUTZ:    SETR     R8=0
         STREGI   R8,R15            ;WRITE OUT A ZERO...NO EDGE
         JMP      NEXT1

NEXT:    SKIPGE   R11=R11-R15
         JMP      OUTZ
         SET      R15=R15+1

NEXT1:   DBNZ     R12,CHKO          ;REPEAT TILL ALL PIXELS TESTED

;        JMP      NRMCND            ;GO TO NORMALIZE THE RESULT DATA, IF FCB CONTROL
                                    ;  BIT INDICATES SUCH REQUIREMENT.
                                    ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR"
                                    ;  FOLLOWED BY AN "RTN".
         RTN
```

```
        END
```

```
        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.
;-----------------------------------------------------------------------
;
;       PROGRAM:        HSTFNC: EDGE DETECTION FOR RAW IMAGE
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 6, 1982
;
;       AUTHOR:         CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS FORTRAN-CALLABLE HOST FUNCTION CALLS UP AN AP-BASED
;                       AP FUNCTION IN ORDER TO PERFORM  "EDGE DETECTION"
;       OPERATION BETWEEN THE RESPECTIVE ELEMENTS OF FOUR AP DATA MEMORY DATA BU
FFERS
;       WHICH CONTAIN THE VARIOUS CONVOLUTION RESULTS OF THE LINE IMAGE WITH A M
ASK.
;
;-----------------------------------------------------------------------

        .TITLE  KVZER - HSTFNC: EDGE DETECTION

        .IDENT  /V01/           ;IDENTIFIER FOR THE OBJECT MODULE.
```

```
        .PAGE
;ESTABLISH ASSEMBLY AND LISTING CONVENTIONS:

        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .DSABL  GBL             ;FLAG NON-EXISTENT-SYMBOL REFERENCES AS ERRORS.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.

        .CSECT  KVZER           ;ESTABLISH A NAMED CSECT.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KVZER

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KEXFCB          ;AP MANAGER'S "FCB EXECUTION" SUBROUTINE.
        .GLOBL  KWAIT           ;AP MANAGER'S WAIT ROUTINE
        .GLOBL  MGRM67          ;AP MANAGER'S "FATAL ERROR #-67" EXIT ROUTINE.
        .GLOBL  COMCTL          ;AP MANAGER'S "FCB CONTROL WORD".

;AP FUNCTION ID'S REFERENCED:

VZER=   ^D810.          ;ID FOR "VERTICAL EDGE DETECTION".


;SYMBOL DEFINITIONS:

        ;NONE

;TERMINOLOGY:

;       FCB -   FUNCTION CONTROL BLOCK, READ BY THE AP EXECUTIVE FROM HOST
;               MEMORY.

        .PAGE
;)+HOST FUNCTION        "KVZER"

;THIS HOST FUNCTION CALLS UP A CORRESPONDING AP FUNCTION IN THE AP400.

;THIS HOST FUNCTION VERSION ASSUMES THAT SOURCE DATA ALREADY RESIDES IN FOUR AP
;DATA MEMORY DATA BUFFERS, AND THAT THE RESULT DATA WILL BE PLACED IN ANOTHER
;AP DATA MEMORY DATA BUFFER.

;THE CORRESPONDING "EDGE DETECTION" AP FUNCTION SHOULD BE REFERENCED FOR FURTHER
;INFORMATION.

;CALL FROM FORTRAN VIA:
```

```
;        SUBROUTINE CALL:   .    CALL    XVZER ( DBIa, DBIb, DBIc, DBId, DBIe,DBI
f )

;        OR INTEGER FUNCTION CALL, AS:    IERR = KVZER ( DBIa, DBIb, DBIc, DBId, D
BIe ) )

;WHERE:

;        DBIa =  ID OF AP DATA BUFFER TO HOLD SOURCE DATA.
;                "DBIa" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;                DBF NEED NOT HAVE BEEN PREVIOUSLY ALLOCATED.
;                IF NOT ALREADY ALLOCATED, DBF WILL BE ALLOCATED; SIZE WILL EQUAL
;                THAT OF SOURCE DATA BUFFERS.
;                IF RESULT DBF WAS PREVIOUSLY ALLOCATED, IT MUST BE OF SIZE EQUAL
;                OR GREATER THAN SOURCE DATA BUFFERS.
;        DBIb =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET
;                "DBIb" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;                DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY.
;                DATA BUFFERS DBIb,DBIc,DBId,DBIe MUST BE OF EQUAL LENGTH.
;        DBIc =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET
;                "DBIc" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;                DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY.
;        DBId =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET
;                SAME RESTRICTIONS AS ABOVE APPLY.
;        DBIe =  ID OF AP DATA BUFFER HOLDING EVEN VERT. CONV RESULTS.
;                SAME RESTRICTIONS AS ABOVE APPLY.


;RETURNS TO FORTRAN WITH:

;        ALL ARGUMENTS RETURNED AS RECEIVED.
;        FUNCTION EXECUTION "IN PROGRESS" OR "COMPLETE", DEPENDING UPON CURRENT
;                AP MANAGER "RETURN" STATUS.
;        IF CALLED AS A FORTRAN FUNCTION, THE VALUE RETURNED WILL BE AS SPECIFIED
;                FOR REGISTER "R0", RETURNED FROM AN ASSEMBLY-LANGUAGE CALL.

;        UPON ERROR, A STANDARD AP MANAGER ERROR EXIT WILL BE TAKEN.

;CALL FROM PDP-11 ASSEMBLY LANGUAGE VIA:

;        A FORTRAN-COMPATIBLE CALL SEQUENCE.

;RETURNS TO CALL+1:              (ALWAYS)

;        ALL CONDITIONS AS DESCRIBED FOR THE FORTRAN FUNCTION CALL FORM, ABOVE.
;        R0 =    STATUS VALUE.  (DEFINED BY AP MANAGER.)
;                "KVZER" DEFINES NO UNIQUE VALUES.
;        R1 =    UNDEFINED.
;        R2 =    UNDEFINED.
;        R5 =    UNDEFINED.


;UPON ERROR, WHEN CALLED FROM FORTRAN OR ASSEMBLY LANGUAGE:
```

```
;       IF A FATAL ERROR OCCURS DURING EXECUTION OF THIS HOST FUNCTION OR DURING
;       EXECUTION OF A ROUTINE WHICH IT (IN TURN) CALLS (SUCH AS THE AP MANAGER
;       OR AP DRIVER), THE AP MANAGER'S FATAL ERROR EXIT ROUTINE WILL BE CALLED.
;)-
        .PAGE
KVZER:

        CMPB    (R5), #6            ;CHECK FOR PROPER NUMBER OF ARGUMENTS.
        BNE     ERRORX              ;IF NOT CORRECT NUMBER, HANDLE AS A FATAL ERROR.

        TST     (R5)+               ;STEP POINTER AHEAD TO FIRST ARGUMENT ADDRESS.

        TST     FCBDON              ;TEST FOR COMPLETION OF A PREVIOUS OPERATION.
        BNE     1$                  ;A ZERO "DONE" FLAG INDICATES PREVIOUS OPERATION
                                    ;  STILL IN PROGRESS.
        JSR     PC,KWAIT                    ;WAIT FOR THE AP TO FINISH PROCESSING

1$:     CLR     FCBDON              ;REINITIALIZE THE "DONE" FLAG.

        MOV     COMCTL, FCBCTL      ;RETRIEVE AP MANAGER'S COMMON CONTROL WORD IN
                                    ;  ORDER TO UTILIZE CURRENTLY-SELECTED OPTIONS.
                                    ;  PLACE IT IN FCB'S CONTROL WORD.

        MOV     @(R5)+, FCBARL      ;MOVE SOURCE DATA BUFFER ID "A" INTO FCB
                                    ;  ARGUMENT LIST.
                                    ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.


        MOV     @(R5)+, FCBARL+4            ;MOVE SOURCE DATA BUFFER ID "B" INTO FCB
                                    ;  ARGUMENT LIST.
                                    ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+^D8          ;MOVE SOURCE DATA BUFFER ID "C" INTO FCB
                                    ;  ARGUMENT LIST.

        MOV     @(R5)+,FCBARL+^D12          ;MOVE SOURCE DATA BUFFER ID "D" INTO FCB
                                    ;  ARGUMENT LIST.
        MOV     @(R5)+,FCBARL+^D16          ;MOVE SOURCE DATA BUFFER ID "E" INTO FCB
                                    ;  ARGUMENT LIST.
        MOV     @(R5)+,FCBARL+^D20          ;MOVE SOURCE DATA BUFFER ID "F" INTO FCB
                                    ;  ARGUMENT LIST.
                                    ;  (INCREMENTING R5, ALTHOUGH UNNECESSARY, SAVES
                                    ;  EXECUTION TIME AND ONE MEMORY WORD.)
        MOV     #MGRARG, R5         ;SET UP ADDRESS OF ARGUMENT LIST FOR CALL TO AP
                                    ;  MANAGER.
        JMP     KEXFCB              ;CALL UP THE AP MANAGER TO PROCESS THE FCB.
                                    ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR",
                                    ;  FOLLOWED BY AN "RTS  PC".
                                    ;  "KEXFCB" WILL RETURN ITS STATUS VALUE IN
                                    ;  PDP-11 REGISTER R0 AS WELL AS IN LOCATION
                                    ;  "STATUS".
MGRARG: BR      2$                  ;BRANCH AROUND ARGUMENT LIST. (THIS INSTRUCTION
                                    ;  PROVIDES "NUMBER OF ARGUMENTS" COUNT FOR AP
                                    ;  MANAGER; THE BRANCH IS NEVER ACTUALLY TAKEN.)
        .WORD   FCBBLK              ;ADDRESS OF FCB.
        .WORD   STATUS              ;ADDRESS FOR RETURNED STATUS.
```

```
2$:     ;THIS LABEL MARKS THE END OF THE ARGUMENT LIST.

ERRORX: JMP     MGRM67          ;TAKE AN AP MANAGER STANDARD FATAL ERROR EXIT.
                                ;  RETURN STATUS CODE -67 TO INDICATE "IMPROPER
                                ;  NUMBER OF ARGUMENTS IN PARAMETER LIST".

STATUS: .WORD   0               ;TEMPORARY STORAGE LOCATION FOR RETURNED AP
                                ;  MANAGER STATUS.

        .PAGE
;FUNCTION CONTROL BLOCK:

FCBBLK.

FCBID:  .WORD   VZER            ;ID OF THE AP FUNCTION.
FCBCTL: .WORD   0               ;CONTROL WORD.
FCBDON: .WORD   1               ;DONE FLAG.  INITIALIZED TO "DONE" STATE.
FCBLNK: .WORD   0               ;(HIGH-ORDER.)  HOST MEMORY ADDRESS LINK TO NEXT
        .WORD   0               ;(LOW-ORDER.)   FCB IN HOST MEMORY.  (NONE.)

FCBPLT: .WORD   1               ;FCB PARAMETER LIST TYPE.  (DATA BUFFER ID'S.)
FCBNRG: .WORD   6               ;NUMBER OF ENTRIES IN ARGUMENT LIST.
FCBLEN: .WORD   ^D12            ;LENGTH OF ARGUMENT LIST IN HOST MEMORY WORDS.

FCBARL: .WORD   0               ;RESULT DATA BUFFER ID "A" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "B" ARGUMENT.


        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "C" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "D" ARGUMENT
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "E" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0
        .WORD   0

        .END
```

```
;--------------------------------------------------------------------------------
;
;        PROGRAM:        APFNC: EDGE DETECTION
;
;        PART NUMBER:
;
;        VERSION DATE:   SEPTEMBER 6, 1982
;
;        AUTHORS:        CHETANA BUCH
;
;        HISTORY:
;
;        DESCRIPTION:    THIS AP-BASED AP FUNCTION PERFORMS AN EDGE DETECTION BY
;                        BASICALLY DETECTING A ZERO CROSSING IN THE CONVOLVED RES
ULTS
;        OF THE LINE OF RAW IMAGE DATA. ODD AND EVEN MASKS ARE USED BOTH HORIZONT
ALLY
;        AND VERTICALLY ON THE IMAGE DATA RESULTING IN FOUR DATA BUFFERS WHICH
;        HAVE TO BE STUDIED FOR THE EDGE DETECTION.
;        THE RESULT BUFFER CONTAINS A CODED WORD FOR EACH PIXEL.
;
;        THIS AP FUNCTION IS NORMALLY CALLED UP BY THE AP EXECUTIVE, WHICH
;        RETRIEVES THIS AP FUNCTION'S ID NUMBER FROM A FUNCTION CONTROL BLOCK
;        READ FROM HOST MEMORY.
;
;--------------------------------------------------------------------------------



        TITLE   APFNC: EDGE DETECTION

        NAME    QVZER, 001      ;NAME AND VERSION FOR THE OBJECT MODULE.


        PAGE
        RADIX   H               ;DEFAULT TO HEXADECIMAL RADIX.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:         (IGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        ;NONE

;       GENERAL SYMBOLS

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE
```

```
;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:          (EGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        EGLOBL  PLCHK1, FTLABT, PLDBF, NRMCND

;       GENERAL SYMBOLS:

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE


;SYMBOL DEFINITIONS:

        ;NONE




;TERMINOLOGY:

        ;NONE


        PAGE
        PMORG           ;START OF RELOCATABLE CODE IN PROGRAM MEMORY.

;)+AP FUNCTION  "OVZER"

; This AP Function performs an edge detection. This is actually a zero crossing
; detection scheme.

; Call with:    parameter list type     = 1,    number of arguments     = 6,
;               parameter list length   = 12.

;       word 9  argument #1             = ID of result Data Buffer "A".
;       word 10 argument #1             = Ignored.

;       word 11 argument #2             = ID of source Data Buffer "B".
;       word 12 argument #2             = Ignored.

;       word 13 argument #3             = ID of source Data Buffer "C".
;       word 14 argument #3             = Ignored.

;       word 15 argument #4             = ID of source Data Buffer "D".
```

```
;        word 16 argument #4            = Ignored.

;        word 17 argument #5            = ID of source Data Buffer "E".
;        word 17 argument #5            = Ignored.

; Exits to AP Executive's "Fatal Abort" Service:

;        If an error is found by AP Service Subroutine 'PLDBF' or 'PLCHK1'.
;)-


        PAGE
;DEFINITION OF THE FUNCTION ID FOR THE AP EXECUTIVE FUNCTION TABLE:

        FUNC    %D810, QVZER     ;FUNCTION ID AND ENTRY POINT NAME.


QVZER:
                                 ;SET UP FOR CALL TO PLCHK1
        SETR    R1=1             ;PARAMETER DESCRIPTOR TYPE
        SETR    R2=6             ;# OF ARGUMENTS
        SETR    R3=%D12          ;# OF WORDS IN ARG LIST
        JSR     PLCHK1           ;GO CHECK CORRECTNESS OF VALUES IN FCB.

        JMP     FTLABT           ;RETURNS HERE IF ERROR
                                 ;IF OK, RETURNS HERE
        SETR    R15=6            ;




FETCH:  JSR     PLDBF            ;  FIND SOURCE DATA BUFFERS.
        JMP     FTLABT           ;  ALLOCATE RESULT DBF IF NECESSARY.
        SET     R1=R1+1          ;  SET UP ARGUMENTS FOR A FUNCTION ADDR. CALL.
        PUSH    R1               ;  UPON ERROR, EXIT THROUGH AP EXECUTIVE'S
        DBNZ    R15,FETCH        ;    FATAL ABORT ROUTINE.

        POP     R15              ;R15 --> ZER2
        POP     R14              ;R14 --> ZER1
        POP     R13              ;R13 --> VE2
        POP     R12              ;R12 --> VO2
        POP     R11              ;R11 --> VE1
        POP     R10              ;R10 --> VO1
        SETR    R1=%H0F
        SET     R14=R14-1
        SET     R15=R15-1
        STREG   R1,R14,HI
        STREG   R1,R15,HI
        SETR    R1=0
        STREGI  R1,R14,LO
        STREGI  R1,R15,LO

START:  LDREGI  R3,R10           ;GET FIRST VO1 VALUE
        LDREGI  R4,R12           ;GET FIRST VO2 VALUE

        SETR    R8=0             ;FLAG FOR EVEN/ODD
TEST:   SETR    R9=0             ;FLAG FOR SIGN
```

```
         SKIPGE  R3=R3
         JMP     NEG
         SKIPGE  R4=R4
         JMP     ZERO
CHK:     SKIPLT  R8=R8           ;CHECK IF EVEN OR ODD
         JMP     CHKE            ;IF ODD, CHK EVEN
FINISH:  SET     R14=R14+1       ;ELSE UPDATE POINTERS
         SET     R15=R15+1
NEXT:    DBNZ    R2,START        ;AND CHECK NEXT VALUE

         JMP     NRMCND          ;GO TO NORMALIZE THE RESULT DATA, IF FCB CONTROL
                                 ;  BIT INDICATES SUCH REQUIREMENT.
                                 ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR"
                                 ;  FOLLOWED BY AN "RTN".
         RTN

CHKE:    LDREGI  R3,R11          ;FETCH VE1
         LDREGI  R4,R13          ;FETCH VE2
         SETR    R8=-1           ;FLAG EVEN DATA
         JMP     TEST

NEG      SKIPGE  R4=R4
         JMP     CHK             ;NO SIGN CHANGE
         SKIPLT  R8=R8           ;CHECK FO NEG ODD
         JMP     CHKE
         SETR    R9=-1           ;MARK FOR NEGATIVE #




ZERO:    SKIPLT  R9=R9           ;CHK SIGN
         JMP     NEGNO
         SKIPLT  R3=R3+R4        ;R3 IS -VE ;R4 IS +VE
         JMP     EVEP
A20:     SKIPGE  R8=R8
         JMP     EVEN
         SETR    R9=0
CODE:    LDREG   R5,R13          ;GET CORRES EVEN STRENGTHS
         LDREG   R6,R11
         SKIPGE  R5=R5
         SETR    R5=0
         SKIPGE  R6=R6
         SETR    R6=0
         SET     R7=R6
         SKIPGE  R6=R6-R5        ;R7 IS MAX [R5,R6]
         SET     R7=R5
         SKIPNE  R7=R7'OR'R7
         JMP     CHKE            ;-VE VALUE NOT VALID FOR ODD
         SETR    R6=%D600        ;NOISE THRESHOLD
         SKIPLT  R6=R6-R7
         JMP     CHKE            ;NOISE
         SETR    R6=%H37         ;VERT ODD PIXEL
         SKIPLT  R9=R9
         JMP     OUT
         STREGI  R6,R14
         SET     R15=R15+1
         JMP     DONE
```

```
OUT:    STREGI   R6,R15
        SET      R14=R14+1
DONE:   SET      R11=R11+1
        SET      R13=R13+1
        JMP      NEXT
A10:    SKIPGE   R8=R8
        JMP      EVEP
        SETR     R9=-1
        JMP      CODE

EVEN:   SET      R6=R12-1
        LDREG    R5,R6
        SKIPGE   R5=R5
        SET      R5='COMP'R5
        SETR     R6=%D1000
        SKIPLT   R6=R6-R5
        JMP      FINISH
        SETR     R6=%H36
        STREGI   R6,R15
        SET      R14=R14+1
        JMP      NEXT

EVEP:   SET      R6=R10-1
        LDREG    R5,R6
        SKIPGE   R5=R5
        SET      R5='COMP'R5
        SETR     R6=%D1000



        SKIPLT   R6=R6-R5
        JMP      FINISH
        SETR     R6=%H36
        LDREG    R7,R14
        SKIPNE   R7=R7'XOR'%H37
        JMP      FINISH
        STREGI   R6,R14
        SET      R15=R15+1
        JMP      NEXT

NEGNO:  SKIPGE   R3=R3+R4
        JMP      A10
        JMP      A20


        END
```

```
        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.
;-----------------------------------------------------------------------
;
;       PROGRAM:        HSTFNC: BYTE- UNPACKIG
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 7, 1982
;
;       AUTHOR:         CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS FORTRAN-CALLABLE HOST FUNCTION CALLS UP AN AP-BASED
;                       AP FUNCTION IN ORDER TO PERFORM  "UNPACKING"
;
;-----------------------------------------------------------------------

        .TITLE  KUPAK - HSTFNC: UNPACK DATA FROM BYTE TO WORD FORMAT

        .IDENT  /V01/           ;IDENTIFIER FOR THE OBJECT MODULE.

        .PAGE
;ESTABLISH ASSEMBLY AND LISTING CONVENTIONS:




        .NLIST  TTM             ;PRODUCE LISTING IN WIDE STYLE.
        .DSABL  GBL             ;FLAG NON-EXISTENT-SYMBOL REFERENCES AS ERRORS.
        .ENABL  LC              ;RETAIN LOWER-CASE CHARACTERS AS SUCH.

        .CSECT  KUPAK           ;ESTABLISH A NAMED CSECT.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KUPAK

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KEXFCB          ;AP MANAGER'S "FCB EXECUTION" SUBROUTINE.
        .GLOBL  KWAIT           ;AP MANAGER'S WAIT ROUTINE
        .GLOBL  MGRM67          ;AP MANAGER'S "FATAL ERROR #-67" EXIT ROUTINE.
        .GLOBL  COMCTL          ;AP MANAGER'S "FCB CONTROL WORD".

;AP FUNCTION ID'S REFERENCED:

UPAK=   AD814.          ;ID FOR "UNPACKING".

;SYMBOL DEFINITIONS:

        ;NONE
```

```
;TERMINOLOGY:

;       FCB -   FUNCTION CONTROL BLOCK, READ BY THE AP EXECUTIVE FROM HOST
;               MEMORY.

        .PAGE
;;+HOST FUNCTION          "KUPAK"

;THIS HOST FUNCTION CALLS UP A CORRESPONDING AP FUNCTION IN THE AP400.

;THIS HOST FUNCTION VERSION ASSUMES THAT SOURCE DATA ALREADY RESIDES IN ONE AP
;DATA MEMORY DATA BUFFERS, AND THAT THE RESULT DATA WILL BE PLACED IN RESULT
;AP DATA MEMORY DATA BUFFER WHICH WILL HAVE TWICE THE SIZE OF THE SOURCE.

;THE CORRESPONDING "UNPACKING" AP FUNCTION SHOULD BE REFERENCED FOR FURTHER
;INFORMATION.

;CALL FROM FORTRAN VIA:

;       SUBROUTINE CALL:        CALL    KUPAK ( DBIa, DBIb )

;       OR INTEGER FUNCTION CALL, AS:   IERR = KUPAK ( DBIa, DBIb )

;WHERE:

;       DBIa =  ID OF AP DATA BUFFER TO HOLD RESULT DATA
;               "DBIa" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               THIS BUFFER WILL BE TWICE THE SIZE OF THE SOURCE BUFFER.
;               DBF SHOULD HAVE BEEN PREVIOUSLY ALLOCATED.
;       DBIb =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET.
;               "DBIb" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.

;RETURNS TO FORTRAN WITH:

;       ALL ARGUMENTS RETURNED AS RECEIVED.
;       FUNCTION EXECUTION "IN PROGRESS" OR "COMPLETE", DEPENDING UPON CURRENT
;               AP MANAGER "RETURN" STATUS.
;       IF CALLED AS A FORTRAN FUNCTION, THE VALUE RETURNED WILL BE AS SPECIFIED
;               FOR REGISTER "R0", RETURNED FROM AN ASSEMBLY-LANGUAGE CALL.

;       UPON ERROR, A STANDARD AP MANAGER ERROR EXIT WILL BE TAKEN.

;CALL FROM PDP-11 ASSEMBLY LANGUAGE VIA:

;       A FORTRAN-COMPATIBLE CALL SEQUENCE.

;RETURNS TO CALL+1:             (ALWAYS)

;       ALL CONDITIONS AS DESCRIBED FOR THE FORTRAN FUNCTION CALL FORM, ABOVE.
```

```
;        R0 =    STATUS VALUE.  (DEFINED BY AP MANAGER.)
;                "KUPAK" DEFINES NO UNIQUE VALUES.
;        R1 =    UNDEFINED.
;        R2 =    UNDEFINED.
;        R5 =    UNDEFINED.

;UPON ERROR, WHEN CALLED FROM FORTRAN OR ASSEMBLY LANGUAGE:

;        IF A FATAL ERROR OCCURS DURING EXECUTION OF THIS HOST FUNCTION OR DURING
;        EXECUTION OF A ROUTINE WHICH IT (IN TURN) CALLS (SUCH AS THE AP MANAGER
;        OR AP DRIVER), THE AP MANAGER'S FATAL ERROR EXIT ROUTINE WILL BE CALLED
;)-
        .PAGE
KUPAK.

        CMPB    (R5), #2            ;CHECK FOR PROPER NUMBER OF ARGUMENTS.
        BNE     ERRORX              ;IF NOT CORRECT NUMBER, HANDLE AS A FATAL ERROR.

        TST     (R5)+               ;STEP POINTER AHEAD TO FIRST ARGUMENT ADDRESS.

        TST     FCBDON              ;TEST FOR COMPLETION OF A PREVIOUS OPERATION.
        BNE     1$                  ;A ZERO "DONE" FLAG INDICATES PREVIOUS OPERATION
                                    ; STILL IN PROGRESS.
        JSR     PC,KWAIT                   ;WAIT FOR THE AP TO FINISH PROCESSING

1$.     CLR     FCBDON              ;REINITIALIZE THE "DONE" FLAG.




        MOV     COMCTL, FCBCTL      ;RETRIEVE AP MANAGER'S COMMON CONTROL WORD IN
                                    ; ORDER TO UTILIZE CURRENTLY-SELECTED OPTIONS.
                                    ; PLACE IT IN FCB'S CONTROL WORD.

        MOV     @(R5)+, FCBARL      ;MOVE RESULT DATA BUFFER ID "A" INTO FCB
                                    ; ARGUMENT LIST.
                                    ; STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+4           ;MOVE SOURCE DATA BUFFER ID "B" INTO FCB
                                    ; ARGUMENT LIST.
                                    ; STEP HOST MEMORY ADDRESS POINTER AHEAD.
                                    ; (INCREMENTING R5, ALTHOUGH UNNECESSARY, SAVES
                                    ; EXECUTION TIME AND ONE MEMORY WORD.)
        MOV     #MGRARG, R5         ;SET UP ADDRESS OF ARGUMENT LIST FOR CALL TO AP
                                    ; MANAGER.
        JMP     KEXFCB              ;CALL UP THE AP MANAGER TO PROCESS THE FCB.
                                    ; A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR",
                                    ; FOLLOWED BY AN "RTS  PC".
                                    ; "KEXFCB" WILL RETURN ITS STATUS VALUE IN
                                    ; PDP-11 REGISTER R0 AS WELL AS IN LOCATION
                                    ; "STATUS".
MGRARG. BR      2$                  ;BRANCH AROUND ARGUMENT LIST.  (THIS INSTRUCTION
                                    ; PROVIDES "NUMBER OF ARGUMENTS" COUNT FOR AP
                                    ; MANAGER; THE BRANCH IS NEVER ACTUALLY TAKEN.)
        .WORD   FCBBLK              ;ADDRESS OF FCB.
        .WORD   STATUS              ;ADDRESS FOR RETURNED STATUS.
2$:     ;THIS LABEL MARKS THE END OF THE ARGUMENT LIST.
```

```
ERRORX: JMP     MGRM67          ;TAKE AN AP MANAGER STANDARD FATAL ERROR EXIT.
                                ;  RETURN STATUS CODE -67 TO INDICATE "IMPROPER
                                ;  NUMBER OF ARGUMENTS IN PARAMETER LIST".

STATUS: .WORD   0               ;TEMPORARY STORAGE LOCATION FOR RETURNED AP
                                ;  MANAGER STATUS.

        .PAGE
;FUNCTION CONTROL BLOCK:

FCBBLK:

FCBID:  .WORD   UPAK            ;ID OF THE AP FUNCTION.
FCBCTL: .WORD   0               ;CONTROL WORD.
FCBDON: .WORD   1               ;DONE FLAG.  INITIALIZED TO "DONE" STATE.
FCBLNK: .WORD   0               ;(HIGH-ORDER.)  HOST MEMORY ADDRESS LINK TO NEXT
        .WORD   0               ;(LOW-ORDER.)   FCB IN HOST MEMORY.  (NONE.)

FCBPLT: .WORD   1               ;FCB PARAMETER LIST TYPE.  (DATA BUFFER ID'S.)
FCBNRG: .WORD   2               ;NUMBER OF ENTRIES IN ARGUMENT LIST.
FCBLEN: .WORD   4               ;LENGTH OF ARGUMENT LIST IN HOST MEMORY WORDS

FCBARL: .WORD   0               ;RESULT DATA BUFFER ID "A" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .WORD   0               ;SOURCE DATA BUFFER ID "B" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.



        .END
```

```
;------------------------------------------------------------------------
;
;       PROGRAM:        APFNC: UNPACKING OF DATA
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 7, 1982
;
;       AUTHORS:        CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS AP-BASED AP FUNCTION PERFORMS UNPACKING OF DATA IN
;                       THE AP FROM BYTE FORMAT TO WORD FORMAT.
;
;       THIS AP FUNCTION IS NORMALLY CALLED UP BY THE AP EXECUTIVE, WHICH
;       RETRIEVES THIS AP FUNCTION'S ID NUMBER FROM A FUNCTION CONTROL BLOCK
;       READ FROM HOST MEMORY.
;
;------------------------------------------------------------------------

        TITLE   APFNC: UNPACK DATA

        NAME    QUPAK, 001      ;NAME AND VERSION FOR THE OBJECT MODULE.




        PAGE
        RADIX   H               ;DEFAULT TO HEXADECIMAL RADIX.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:         (IGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        ;NONE

;       GENERAL SYMBOLS

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:         (EGLOBL)

;       ENTRY POINTS:
```

```
        ;NONE
;       SUBROUTINES:
        EGLOBL  PLCHK1, FTLABT, PLDBF, NRMCND
;       GENERAL SYMBOLS:
        ;NONE
;       DATA MEMORY LABELS:
        ;NONE

;SYMBOL DEFINITIONS:
        ;NONE

;TERMINOLOGY:
        ;NONE

        PAGE

        PMORG           ;START OF RELOCATABLE CODE IN PROGRAM MEMORY.

;)+AP FUNCTION  "QUPAK"

; This AP Function performs unpacking of data from a source buffer.The resulting
; buffer will be twice the size of the the source buffer.

; Call with:    parameter list type     = 1,    number of arguments    = 2,
;               parameter list length   = 4.

;       word 9  argument #1             = ID of result Data Buffer "A".
;       word 10 argument #1             = Ignored.

;       word 11 argument #2             = ID of source Data Buffer "B"
;       word 12 argument #2             = Ignored.

; Exits to AP Executive's "Fatal Abort" Service:

;       If an error is found by AP Service Subroutine 'PLDBF' of 'PLCHK1'.
;)-

       PAGE
;DEFINITION OF THE FUNCTION ID FOR THE AP EXECUTIVE FUNCTION TABLE:

       FUNC    %D814, QUPAK    ;FUNCTION ID AND ENTRY POINT NAME.
```

```
OUPAK:
                                     ;SET UP FOR CALL TO PLCHK1
         SETR     R1=1               ;PARAMETER DESCRIPTOR TYPE
         SETR     R2=2               ;# OF ARGUMENTS
         SETR     R3=4               ;# OF WORDS IN ARG LIST
         JSR      PLCHK1             ;GO CHECK CORRECTNESS OF VALUES IN FCB.

         JMP      FTLABT             ;RETURNS HERE IF ERROR
                                     ;IF OK, RETURNS HERE
         JSR      PLDBF
         JMP      FTLABT
         SET      R15=R1             ;POINTS TO RESULT BUFFER
         JSR      PLDBF
         JMP      FTLABT             ;R1 POINTS TO SECOND SOURCE BUFFER

         LDREGI   R10,R1             ;FETCH BEX/NSN
         STREGI   R10,R15

NEXT.    LDREGI   R3,R1              ;FETCH NEXT WORD
         SET      R4=R3
         SETR     R6=%HFF            ;SET MASK
         SET      R3=R3'AND'R6       ;R3-->LS BYTE WORD
         SETR     R5=8
SHIFT:   SET      R4=R4/2            ;SHIFT RIGHT
         DBNZ     R5,SHIFT


         SET      R4=R4'AND'R6       ;R4-->MS BYTE WORD
         STREGI   R3,R15
         STREGI   R4,R15
         DBNZ     R2,NEXT

;        JMP      NRMCND             ;GO TO NORMALIZE THE RESULT DATA, IF FCB CONTROL
                                     ;  BIT INDICATES SUCH REQUIREMENT.
                                     ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR"
                                     ;  FOLLOWED BY AN "RTN".
         RTN

         END
```

```
        .NLIST  TTM                 ;PRODUCE LISTING IN WIDE STYLE.
        .ENABL  LC                  ;RETAIN LOWER-CASE CHARACTERS AS SUCH.
;-------------------------------------------------------------------------------
;
;       PROGRAM:        HSTFNC: OR-ING OF TWO DATA BUFFERS
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 7, 1982
;
;       AUTHOR:         CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS FORTRAN-CALLABLE HOST FUNCTION CALLS UP AN AP-BASED
;                       AP FUNCTION IN ORDER TO PERFORM  "OR-ING"
;       OPERATION BETWEEN THE RESPECTIVE ELEMENTS OF TWO AP DATA MEMORY DATA BUF
FERS
;
;-------------------------------------------------------------------------------

        .TITLE  KORDB - HSTFNC: OR TWO DATA BUFFERS

        .IDENT  /V01/               ;IDENTIFIER FOR THE OBJECT MODULE.

        .PAGE



;ESTABLISH ASSEMBLY AND LISTING CONVENTIONS:

        .NLIST  TTM                 ;PRODUCE LISTING IN WIDE STYLE.
        .DSABL  GBL                 ;FLAG NON-EXISTENT-SYMBOL REFERENCES AS ERRORS.
        .ENABL  LC                  ;RETAIN LOWER-CASE CHARACTERS AS SUCH.

        .CSECT  KORDB               ;ESTABLISH A NAMED CSECT.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KORDB

.EXTERNALLY DEFINED GLOBALIZED SYMBOLS:

        .GLOBL  KEXFCB              ;AP MANAGER'S "FCB EXECUTION" SUBROUTINE.
        .GLOBL  KWAIT               ;AP MANAGER'S WAIT ROUTINE
        .GLOBL  MGRM67              ;AP MANAGER'S "FATAL ERROR #-67" EXIT ROUTINE.
        .GLOBL  COMCTL              ;AP MANAGER'S "FCB CONTROL WORD".

;AP FUNCTION ID'S REFERENCED:

ORDB=   ^D812.          ;ID FOR "OR-ING".
```

```
;SYMBOL DEFINITIONS:

        ;NONE

;TERMINOLOGY:

;       FCB -   FUNCTION CONTROL BLOCK, READ BY THE AP EXECUTIVE FROM HOST
;               MEMORY.

        .PAGE
;)+HOST FUNCTION         "KORDB"

;THIS HOST FUNCTION CALLS UP A CORRESPONDING AP FUNCTION IN THE AP400.

;THIS HOST FUNCTION VERSION ASSUMES THAT SOURCE DATA ALREADY RESIDES IN TWO AP
;DATA MEMORY DATA BUFFERS, AND THAT THE RESULT DATA WILL BE PLACED IN ONE OF THE
SE
;AP DATA MEMORY DATA BUFFER.

;THE CORRESPONDING "OR-ING" AP FUNCTION SHOULD BE REFERENCED FOR FURTHER
;INFORMATION.

;CALL FROM FORTRAN VIA:


;       SUBROUTINE CALL:        CALL    KORDB ( DBIa, DBIb )

;       OR INTEGER FUNCTION CALL, AS:   IERR = KORDB ( DBIa, DBIb )

;WHERE:

;       DBIa =  ID OF AP DATA BUFFER TO HOLD RESULT DATA AND CONTAINS SOURCE DAT
A
;               "DBIa" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF NEED NOT HAVE BEEN PREVIOUSLY ALLOCATED.
;               IF NOT ALREADY ALLOCATED, DBF WILL BE ALLOCATED; SIZE WILL EQUAL
;               THAT OF SOURCE DATA BUFFERS.
;               IF RESULT DBF WAS PREVIOUSLY ALLOCATED, IT MUST BE OF SIZE EQUAL
;               OR GREATER THAN SOURCE DATA BUFFERS.
;       DBIb =  ID OF AP DATA BUFFER HOLDING SOURCE DATA SET.
;               "DBIb" MUST BE A SINGLE-WORD INTEGER VARIABLE OR CONSTANT.
;               DBF MUST HAVE BEEN PREVIOUSLY ALLOCATED IN AP DATA MEMORY.
;               DATA BUFFERS DBIb,DBIc,DBId,DBIe MUST BE OF EQUAL LENGTH.

;RETURNS TO FORTRAN WITH:

;       ALL ARGUMENTS RETURNED AS RECEIVED.
;       FUNCTION EXECUTION "IN PROGRESS" OR "COMPLETE", DEPENDING UPON CURRENT
;               AP MANAGER "RETURN" STATUS.
;       IF CALLED AS A FORTRAN FUNCTION, THE VALUE RETURNED WILL BE AS SPECIFIED
;               FOR REGISTER "R0", RETURNED FROM AN ASSEMBLY-LANGUAGE CALL.
```

```
;       UPON ERROR, A STANDARD AP MANAGER ERROR EXIT WILL BE TAKEN.

;CALL FROM PDP-11 ASSEMBLY LANGUAGE VIA:

;       A FORTRAN-COMPATIBLE CALL SEQUENCE.

;RETURNS TO CALL+1:             (ALWAYS)

;       ALL CONDITIONS AS DESCRIBED FOR THE FORTRAN FUNCTION CALL FORM, ABOVE.
;       R0 =    STATUS VALUE.  (DEFINED BY AP MANAGER.)
;               "KORDB" DEFINES NO UNIQUE VALUES.
;       R1 =    UNDEFINED.
;       R2 =    UNDEFINED.
;       R5 =    UNDEFINED.

;UPON ERROR, WHEN CALLED FROM FORTRAN OR ASSEMBLY LANGUAGE:

;       IF A FATAL ERROR OCCURS DURING EXECUTION OF THIS HOST FUNCTION OR DURING
;       EXECUTION OF A ROUTINE WHICH IT (IN TURN) CALLS (SUCH AS THE AP MANAGER
;       OR AP DRIVER), THE AP MANAGER'S FATAL ERROR EXIT ROUTINE WILL BE CALLED.
;)-
        .PAGE
KORDE:

        CMPB    (R5), #2        ;CHECK FOR PROPER NUMBER OF ARGUMENTS.

        BNE     ERRORX          ;IF NOT CORRECT NUMBER, HANDLE AS A FATAL ERROR.

        TST     (R5)+           ;STEP POINTER AHEAD TO FIRST ARGUMENT ADDRESS.

        TST     FCBDON          ;TEST FOR COMPLETION OF A PREVIOUS OPERATION.
        BNE     1$              ;A ZERO "DONE" FLAG INDICATES PREVIOUS OPERATION
                                ;  STILL IN PROGRESS.
        JSR     PC,KWAIT                ;WAIT FOR THE AP TO FINISH PROCESSING

1$:     CLR     FCBDON          ;REINITIALIZE THE "DONE" FLAG.

        MOV     COMCTL, FCBCTL  ;RETRIEVE AP MANAGER'S COMMON CONTROL WORD IN
                                ;  ORDER TO UTILIZE CURRENTLY-SELECTED OPTIONS.
                                ;  PLACE IT IN FCB'S CONTROL WORD.

        MOV     @(R5)+, FCBARL  ;MOVE RESULT DATA BUFFER ID "A" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
        MOV     @(R5)+, FCBARL+4        ;MOVE SOURCE DATA BUFFER ID "B" INTO FCB
                                ;  ARGUMENT LIST.
                                ;  STEP HOST MEMORY ADDRESS POINTER AHEAD.
                                ;  (INCREMENTING R5, ALTHOUGH UNNECESSARY, SAVES
                                ;  EXECUTION TIME AND ONE MEMORY WORD.)
        MOV     #MGRARG, R5     ;SET UP ADDRESS OF ARGUMENT LIST FOR CALL TO AP
                                ;  MANAGER.
        JMP     KEXFCB          ;CALL UP THE AP MANAGER TO PROCESS THE FCB.
                                ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR",
```

```
                                ;  FOLLOWED BY AN "RTS  PC".
                                ;  "KEXFCB" WILL RETURN ITS STATUS VALUE IN
                                ;  PDP-11 REGISTER R0 AS WELL AS IN LOCATION
                                ;  "STATUS".
MGRARG: BR      2$              ;BRANCH AROUND ARGUMENT LIST.  (THIS INSTRUCTION
                                ;  PROVIDES "NUMBER OF ARGUMENTS" COUNT FOR AP
                                ;  MANAGER; THE BRANCH IS NEVER ACTUALLY TAKEN.)
        .WORD   FCBBLK          ;ADDRESS OF FCB.
        .WORD   STATUS          ;ADDRESS FOR RETURNED STATUS.
2$:     ;THIS LABEL MARKS THE END OF THE ARGUMENT LIST.

ERRORX. JMP     MGRM67          ;TAKE AN AP MANAGER STANDARD FATAL ERROR EXIT.
                                ;  RETURN STATUS CODE -67 TO INDICATE "IMPROPER
                                ;  NUMBER OF ARGUMENTS IN PARAMETER LIST".

STATUS. .WORD   0               ;TEMPORARY STORAGE LOCATION FOR RETURNED AP
                                ;  MANAGER STATUS.
        .PAGE
;FUNCTION CONTROL BLOCK:

FCBBLK:

FCBID:  .WORD   ORDB            ;ID OF THE AP FUNCTION.
FCBCTL: .WORD   0               ;CONTROL WORD.
FCBDON: .WORD   1               ;DONE FLAG.  INITIALIZED TO "DONE" STATE.
FCBLNK: .WORD   0               ;(HIGH-ORDER.)  HOST MEMORY ADDRESS LINK TO NEXT
        .WORD   0               ;(LOW-ORDER.)   FCB IN HOST MEMORY.  (NONE.)




FCBPLT: .WORD   1               ;FCB PARAMETER LIST TYPE.  (DATA BUFFER ID'S.)
FCBNRG. .WORD   2               ;NUMBER OF ENTRIES IN ARGUMENT LIST.
FCBLEN: .WORD   4               ;LENGTH OF ARGUMENT LIST IN HOST MEMORY WORDS.

FCBARL: .WORD   0               ;RESULT DATA BUFFER ID "A" ARGUMENT.
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0

        .WORD   0               ;SOURCE DATA BUFFER ID "B" ARGUMENT
        .WORD   0               ;  FIRST WORD = DBF ID; SECOND WORD = 0.

        .END
```

```
;---------------------------------------------------------------------------
;
;       PROGRAM:        APFNC: OR-ING TWO DATA BUFFERS
;
;       PART NUMBER:
;
;       VERSION DATE:   SEPTEMBER 7, 1982
;
;       AUTHORS:        CHETANA BUCH
;
;       HISTORY:
;
;       DESCRIPTION:    THIS AP-BASED AP FUNCTION PERFORMS A LOGICAL 'OR' BETWEE
N
;                       EACH ELEMENT OF TWO DATA BUFFERS.
;
;       THIS AP FUNCTION IS NORMALLY CALLED UP BY THE AP EXECUTIVE, WHICH
;       RETRIEVES THIS AP FUNCTION'S ID NUMBER FROM A FUNCTION CONTROL BLOCK
;       READ FROM HOST MEMORY.
;
;---------------------------------------------------------------------------

        TITLE   APFNC: LOGICL-OR

        NAME    QORDB, 001      ;NAME AND VERSION FOR THE OBJECT MODULE.




        PAGE
        RADIX   H               ;DEFAULT TO HEXADECIMAL RADIX.

;INTERNALLY DEFINED GLOBALIZED SYMBOLS:          (IGLOBL)

;       ENTRY POINTS:

        ;NONE

;       SUBROUTINES:

        ;NONE.

;       GENERAL SYMBOLS

        ;NONE

;       DATA MEMORY LABELS:

        ;NONE

;EXTERNALLY DEFINED GLOBALIZED SYMBOLS:          (EGLOBL)

;       ENTRY POINTS:
```

```
        ;NONE
;       SUBROUTINES:
        EGLOBL  PLCHK1, FTLABT, PLDBF, NRMCND
;       GENERAL SYMBOLS:
        ;NONE
;       DATA MEMORY LABELS:
        ;NONE

;SYMBOL DEFINITIONS
        ;NONE

;TERMINOLOGY:
        ;NONE


        PAGE
        PMORG           ;START OF RELOCATABLE CODE IN PROGRAM MEMORY.

;)+AP FUNCTION  "OORDB"
; This AP Function performs a logical or between two data buffers.
; Call with:    parameter list type     = 1,    number of arguments     = 2,
;               parameter list length   = 4.
;       word 9  argument #1             = ID of source and result Data Buffer "A
".
;       word 10 argument #1             = Ignored.
;       word 11 argument #2             = ID of source Data Buffer "B".
;       word 12 argument #2             = Ignored.
; Exits to AP Executive's "Fatal Abort" Service:
;       If an error is found by AP Service Subroutine 'PLDBF' or 'PLCHK1'.
;)-

        PAGE
;DEFINITION OF THE FUNCTION ID FOR THE AP EXECUTIVE FUNCTION TABLE:
```

```
        FUNC    %D812, QORDB   ;FUNCTION ID AND ENTRY POINT NAME.

QORDB:
                               ;SET UP FOR CALL TO PLCHK1
        SETR    R1=1           ;PARAMETER DESCRIPTOR TYPE
        SETR    R2=2           ;# OF ARGUMENTS
        SETR    R3=4           ;# OF WORDS IN ARG LIST
        JSR     PLCHK1         ;GO CHECK CORRECTNESS OF VALUES IN FCB.

        JMP     FTLABT         ;RETURNS HERE IF ERROR
                               ;IF OK, RETURNS HERE
        JSR     PLDBF
        JMP     FTLABT
        SET     R11=R1         ;POINTS TO SOURCE/RESULT BUFFER
        JSR     PLDBF
        JMP     FTLABT         ;R1 POINTS TO SECOND SOURCE BUFFER
        SET     R2=R2+1        ;LENGTH ( + FOR BEX/NSN OF BUFFER )

OR:     LDREG   R14,R11        ;FETCH DATA FROM FIRST SOURCE BUFFER
        LDREGI  R15,R1         ;ALSO FROM SECOND
        SET     R14=R14'OR'R15 ;LOGICAL-OR
        STREGI  R14,R11        ;STORE RESULT BACK IN SOURCE 1 BUFFER
        DBNZ    R2,OR

;       JMP     NRMCND         ;GO TO NORMALIZE THE RESULT DATA, IF FCE CONTROL
                               ;  BIT INDICATES SUCH REQUIREMENT.


                               ;  A DIRECT BRANCH IS THE EQUIVALENT OF A "JSR"
                               ;  FOLLOWED BY AN "RTN"
        RTN

        END
```

STAGE MOVEMENT AND IMAGE ACQUISITION

```
(* ********************************************************************
      STAGET.MC - THIS MODULE LOADS ALL OF THE MODULES USED IN "STAGET"
 ******************************************************************** *)

ext     PDPID
ext     mixlib
ext     DMISC
ext     VADD
ext     STGREG
ext     22BADDR
ext     STGGRB
ext     STGCOM
ext     CMOTOR
ext     [5,1]INSPLAN
ext     SQROOT

save    STAGE
```

```
        (*       INTEGER VECTOR ROUTINES         *)

IF ( LOOKUP ( 'VADD ) )
        PRINT "VADD ALREADY LOADED"
        ENDFILE
ENDIF

.MAC

; VADD  -- ADD TWO VECTORS OF ANY LENGTH
; CALL:           VADD ( IVEC1 , IVEC2 , LENGTH )
; CALCULATES:     IVEC2 ( I ) := IVEC1 ( I ) + IVEC2 ( I )

ENTRY VADD
        MOV (MSP)+ , R0           ; GET THE LENGTH
        MOV (MSP)+ , R1           ; GET THE ADDRESS OF IVEC2
        MOV (MSP)+ , R2           ; GET THE ADDRESS OF IVEC1

0$:     ADD (R2)+ , (R1)+         ; ADD THE COMPONENTS AND INCREMENT THE POINTERS
        DEC R0                    ; DECREMENT COUNTER AND CHECK DONE
        BNE 0$

        NEXT

; VSUB  -- SUBTRACT TWO VECTORS
; CALL:           VSUB ( IVEC1 , IVEC2 , LENGTH )
; CALCULATES:     IVEC2 ( I ) := IVEC2 ( I ) - IVEC1 ( I )
```

```
ENTRY VSUB
        MOV (MSP)+ , R0          ; LENGTH
        MOV (MSP)+ , R1          ; ADDRESS OF IVEC2
        MOV (MSP)+ , R2          ; ADDRESS OF IVEC1

1$:     SUB (R2)+ , (R1)+        ; SUBTRACT COMPONENTS AND INCREMENT POINTERS
        DEC R0                   ; DECREMENT COUNTER AND CHECK DONE
        BNE 1$

        NEXT

; VMASK  -- MASK ALL THE ELEMENTS IN A VECTOR
; CALL:          VMASK ( IVEC , MASK , LENGTH )
: CALCULATES:    IVEC ( I ) := IVEC ( I ) and MASK

ENTRY VMASK
        MOV (MSP)+ , R0          ; LENGTH
        MOV (MSP)+ , R1          ; MASK
        MOV (MSP)+ , R2          ; ADDRESS OF IVEC
        COM R1                   ; COMPLEMENT THE MASK. BIC DOES- ~SRC and DST

2$:     BIC R1 , (R2)+           ; AND THE COMPONENT WITH MASK AND INCR POINTER
        DEC R0                   ; DECREMENT COUNTER AND CHECK DONE
        BNE 2$

        NEXT



; VMAX  -- FIND THE INDEX OF THE MAXIMUM VALUE IN A VECTOR
; CALL:          INDEX := VMAX ( IVEC , LENGTH )

ENTRY VMAX
        MOV (MSP)+ , R0          ; LENGTH
        MOV (MSP) , R1           ; ADDRESS OF IVEC. KEEP ON STACK
        MOV R1 , R2              ; ASSUME 1ST ELEMENT. SAVE ADDRESS OF 1ST ELT
        MOV (R1) , R3            ; SAVE VALUE OF 1ST ELEMENT

3$:     CMP R3 , (R1)            ; COMPARE CURRENT MAX TO COMPONENT
        BGE 4$                   ; IF MAX > COMPONENT LEAVE IT
        MOV (R1) , R3            ; NEW MAX VALUE
        MOV R1 , R2              ; NEW ADDR OF MAX VALUE
4$:     ADD # 2 , R1             ; INCREMENT ARRAY INDEX
        DEC R0                   ; DECREMENT COUNTER AND CHECK DONE
        BNE 3$

8$:     SUB (MSP) , R2           ; ENTRY FOR VMIN ALSO. SUBTRACT ADDR OF MAX VAL
        MOV R2 , (MSP)           ; FROM ADDR OF IVEC AND STORE DIFFERENCE ON STAC
K
        ASR (MSP)                ; DIVIDE RESULT BY 2 TO GET NUMBER OF WORDS
        NEXT

; VMIN  -- FIND INDEX OF MINIMUM VALUE IN A VECTOR
; CALL:          INDEX := VMIN ( IVEC , LENGTH )
```

```
ENTRY VMIN
        MOV (MSP)+ , R0          ; LENGTH
        MOV (MSP) , R1           ; ADDR OF IVEC. KEEP ON STACK
        MOV R1 , R2              ; ASSUME 1ST ELT. STORE ADDR OF 1ST ELT
        MOV (R1) , R3            ; STORE VALUE OF 1ST ELT.  CURRENT MAX

5$:     CMP R3 , (R1)            ; COMPARE CURRENT MAX TO COMPONENT
        BLE 6$                   ; IF MAX ) COMPONENT, LEAVE IT
        MOV (R1) , R3            ; NEW MAX VALUE
        MOV R1 , R2              ; NEW MAX VALUE ADDR
6$:     ADD # 2 , R1             ; INCREMENT ARRAY POINTER
        DEC R0                   ; DECREMENT COUNTER AND CHECK DONE
        BNE 5$
        JMP 8$                   ; CALCULATE INDEX AND RETURN

. VSDIV  -- DIVIDE A VECTOR BY A SCALAR
; CALL:          VSDIV ( IVEC , SCALAR , LENGTH )
: CALCULATES:    IVEC ( I ) := IVEC ( I ) / SCALAR

ENTRY VSDIV
        MOV (MSP)+ , R3          ; LENGTH
        MOV (MSP)+ , R2          ; SCALAR

7$:     MOV @ 0 (MSP) , R1       ; ADDR OF IVEC STILL ON STACK. GET ELEMENT
        SXT R0                   ; IF (0 SET HIGH 16 BITS TO -1 ELSE CLEAR THEM
        DIV R2 , R0              ; 32 BIT DIVIDE. QUOTIENT IN R0, REMAINDER IN R1
        MOV R0 , @ 0 (MSP)       ; STORE QUOTIENT BACK IN ELEMENT




        ADD # 2 , (MSP)          ; UPDATE THE ARRAY POINTER
        DEC R3                   ; DECREMENT COUNTER AND CHECK DONE
        BNE 7$

        CLR (MSP)+               ; POP THE ADDRESS OF IVEC OFF STACK
        NEXT

; VSMUL  -- MULTIPLY A VECTOR BY A SCALAR
; CALL:          VSMUL ( IVEC , SCALAR , LENGTH )
; CALCULATES:    IVEC ( I ) := IVEC ( I ) * SCALAR

ENTRY VSMUL
        MOV (MSP)+ , R3          ; LENGTH
        MOV (MSP)+ , R2          ; SCALAR
9$:     MOV @ 0 (MSP) , R1       ; ADDR OF IVEC STILL ON STACK. GET ELEMENT
        MUL R2 , R1              ; MULTIPLY. ONLY 16 BITS SINCE SRC IS R1
        MOV R1 , @ 0 (MSP)       ; PUT RESULT IN ELEMENT
        ADD # 2 , (MSP)          ; UPDATE ARRAY POINTER
        DEC R3                   ; DONE?
        BNE 9$

        CLR (MSP)+               ; POP ADDRESS OF IVEC OFF STACK
        NEXT

.LOCAL                           ; RESET LOCAL SYMBOLS

; VPOS  -- CONVERT ALL NEGATIVE ELEMENTS IN A VECTOR TO POSITIVE
```

```
;           SET ALL POSITIVE ELEMENTS TO IVAL ... IF IVAL=0 LEAVE ALONE
; CALL:           VPOS ( IVEC , IVAL , LENGTH )
; CALCULATES:     IF ( IVEC ( I ) < 0 ) IVEC ( I ) := 0 ;; ENDIF
;                 IF ( IVAL <>0 ) IVEC ( I ) := IVAL ;; ENDIF

ENTRY VPOS
        MOV (MSP)+ , R0            ; LENGTH
        MOV (MSP)+ , R1            ; IVAL
        MOV (MSP)+ , R2            ; ADDRESS OF IVEC
0$:     TST (R2)                   ; TEST THE COMPONENT OF IVEC
        BPL 1$                     ; IF >= 0 GO TO 1$
        CLR (R2)                   ; SET ELEMENT TO 0
        BR 2$                      ; LOOP
1$:     TST R1                     ; TEST IVAL
        BEQ 2$                     ; IF 0 IGNORE
        MOV R1 , (R2)              ; SET ELEMENT TO IVAL
2$:     ADD # 2 , R2               ; UPDATE ARRAY POINTER
        DEC R0                     ; DONE?
        BNE 0$

        NEXT

; VDOT  -- TAKE SCALAR PRODUCT OF TWO VECTORS
; CALL:           PRODUCT := VDOT ( IVEC1 , IVEC2 , LENGTH )
; CALCULATES:     PRODUCT := SUM-OVER-I ( IVEC1 ( I ) * IVEC2 ( I ) )

ENTRY VDOT        INTEGER


        MOV (MSP)+ , R3            ; LENGTH
        MOV (MSP)+ , R2            ; ADDRESS OF IVEC2
        CLR R0                     ; CLEAR SUM

3$:     MOV (R2)+ , R1             ; GET ELT OF IVEC2 AND UPDATE ARRAY POINTER
        MUL @ 0 (MSP) , R1         ; ADDR OF IVEC1 IS STILL ON STACK. GET ELT OF IV
EC1
        ADD R1 , R0                ; ADD COMPONENT PRODUCT TO SUM
        ADD # 2 , (MSP)            ; UPDATE IVEC1 ARRAY POINTER
        DEC R3                     ; DONE
        BNE 3$

        MOV R0 , (MSP)             ; STORE SUM ON STACK
        NEXT
.END
```

```
parameter WCR := 172410k          ; DMA word count register.
parameter BAR := 172412k          ; Bus address register for DMA.
parameter CSR := 172414k          ; Control status register.
parameter DBR := 172416k          ; Data buffer register.

long    PHYADR                    ; Physical (22-bit) address of the buffer.

integer OUTLN ( 256. )


define ATIOPAGE
  with M_IOPAGE
  ATTRG ( "IOPAGE" , 160000X )
end


(*      Wait until DMA operation is complete.  (Monitors BUSY bit.)      *)
define WBUS?Y
  while ( peek ( CSR ) AND 200k )
  repeat
end

define RDLN
        integer BUFF ( 1 ) x0 x1 y0 y1
  PHYADR := 22ADDR ( BUFF )
  poke ( 130000k + X0 - 1 , DBR )
  poke ( 114000k + Y0 , DBR )




  poke ( -- XL / 2 , WCR )
;  poke ( -- ( XL * YL ) , WCR )
  poke ( lsword ( PHYADR ) , BAR )
  poke ( 0 , DBR )
  poke ( msword ( PHYADR ) + 1 , CSR )
end
```

```
integer MOTCH

char buf ( 30 ) TERM ( 5 )
integer bufptr
integer tempr0
.mac

label CMOTAST
        mov     r0 , @ # ptr ( tempr0 )
        mov     @ # ptr ( bufptr ) , r0
        movb    (rp)+ , (r0)

        cmpb    # 15k , (r0)+
        bne     0$
        clrb    (r0)
        mov     # 21. , -(rp)
        mov     # bytewd ( 2 33. ) , -(rp)
        emt     377k
0$:
        mov     @ # ptr ( tempr0 ) , r0
        inc     @ # ptr ( bufptr )
        mov     # bytewd ( 1 115. ) , -(rp)      ; return from ast
        emt     377k
.end

make '$GRBMOT $attach 1410k CMOTAST 0 base NO_OP ; detterm




define GRBMOT
  apush cich
  cich := MOTCH
  $GRBMOT
  apop
end

make 'wtse rsxcall bytewd ( 2 41. )
make 'CLEF$ rsxcall bytewd ( 2 31. )

define bfwrl
        integer buff
  bufptr := buf
  CLEF$ ( 21. )
  wrl ( MOTCH buff )
  wtse ( 21. )
end


parameter CRCHR := 15k

record DEV_REC
        integer DEVNUM
        long    LOLIMIT
        long    UPLIMIT
endrecord
```

```
DEV_REC XAXIS   YAXIS   ZAXIS
with XAXIS
        DEVNUM := ascii 1
with YAXIS
        DEVNUM := ascii 2
with ZAXIS
        DEVNUM := ascii 3

char    BUFF ( 20 )

integer STEPSIZE


define INITMOT
  MOTCH := open ( TERM , 'rwa )
  poke ( 2 , fdb ( MOTCH ) )
  GRBMOT
end


define MOTCOM command
        integer CHAR
  #tyo ( DEVNUM )
  iter cmdcnt
        #tyo ( CHAR )




        nxtarg
  loop
  #tyo ( CRCHR )
  encode ( BUFF )
  bfwrl ( BUFF )
end


; Switches the box on

define BXON
  MOTCOM ascii E
end

; Switches the box off

define BXOFF
   MOTCOM ascii F
end

define JON
   MOTCOM ascii J ascii 1
end

define JOFF
  MOTCOM ascii J ascii 0
end
```

```
; Stop the move

define STP
  MOTCOM ascii S
end

define WBUSYM
   MOTCOM ascii R
end

; Normal mode is single move mode. The motor accelerates
; to the programmed velocity , runs at constant velocity
; for a predetermined period and decelerates and stops when
; total number of counts programmed by position have been sent.

define NORMALMODE
  MOTCOM ascii M , ascii N
end
;
; Continuous mode accelerates the motor to the programmed velocity
; and holds that velocity until stopped.

; define CONTMODE
;  MOTCOM ascii M , ascii C
; end
;




; Alternates the defined move in opposite directions
; until stop is pressed.
; define ALTMODE
;   MOTCOM ascii M , ascii A
; end
;
; This routine reports relative position at the end of the move
;
;
define RELPOS
  #tyo ( DEVNUM )
  #tyo ( ascii P )
  #tyo ( 40k )
  encode ( BUFF )
  BFWRL ( BUFF )
end
;
; Position report back of the motor shaft during move.
; New position at every 4 msec.
 define CURPOS
   MOTCOM ascii W , ascii 2
end
;
; Absolute position relative to the last time the position
; counter was cleared.

define ABSPOS
```

```
   #tyo ( DEVNUM )
   #tyo ( ascii X )
   #tyo ( ascii 1 )
   #tyo ( 40k )
   encode ( BUFF )
   BFWRL ( BUFF )
 ;  MOTCOM ascii X , ascii 1
 end
 ;
 ; Clear the absolute position counter
 define CLEARIND
   MOTCOM ascii X , ascii 0
 end
 ;
 ; Changes the report back ASCII strings into long integers.

 define CONVERT long
   bpoke ( 0 , bp ( bufptr - 1 ) )
   inp := buf
   eol off
   word drop
   word drop
   eol on
   if ( lliteral ( tbuf ) ) endif
   CONVERT := llval
 end



 ; define CONVERT long
 ;  mvzer ( BUFF , 10 )
 ;  rdl ( MOTCH , BUFF ) drop
 ;  if ( lliteral ( BUFF ) ) CONVERT := llval endif
 ; end
 ;
 ; Set up the lower limit for position
 ;  define SETLO
 ;    ABSPOS
 ;    LOLIMIT := CONVERT
 ;  end
 ;
 ; Set up the upper limit for position.
 ;  define SETUP
 ;    ABSPOS
 ;    UPLIMIT := CONVERT
 ;  end
 ;
 ; Set the acceleration of the motor shaft in rps/s


 define ACCEL command
         real N
   if ( cmdcnt ==0 ) N := 0.2 endif
   #tyo ( DEVNUM )
   #tyo ( ascii A )
   mvzer ( BUFF , 10 )
```

```
  print #p ascii 0 , #f 6 3 , N , #n
  #tyo ( CRCHR )
  encode ( BUFF )
;  wrl ( MOTCH , BUFF )
  bfwrl ( BUFF )
;  rdl ( MOTCH , BUFF ) drop
end
;
; Set the delay desired between two commands in seconds.

(*
define TIMEDELAY command
        real N
  if ( cmdcnt ==0 ) N := 1.0 endif
  #tyo ( DEVNUM )
  #tyo ( ascii T )
  mvzer ( BUFF , 10 )
  print #p ascii 0 , #f 6. 3 , N , #n
  #tyo ( CRCHR )
  encode ( BUFF )
;  wrl ( MOTCH , BUFF )
  bfwrl ( BUFF )
,  rdl ( MOTCH , BUFF ) drop
end
*)
;
; Set the velocity of motor in real values




define VELOCITY command"
        real N
  if ( cmdcnt ==0 ) N := 0.1 endif
  #tyo ( DEVNUM )
  #tyo ( ascii V )
  mvzer ( BUFF , 10 )
  print #p ascii 0 , #f 6. 3 , N , #n
  #tyo ( CRCHR )
  encode ( BUFF )
;  wrl ( MOTCH , BUFF )
  bfwrl ( BUFF )
;  rdl ( MOTCH , BUFF ) drop
end

;
; Set the motor position in (+) ve or (-) ve direction
; with respect to the current position in terms of motor
; pulses. 25000 pulses /rev and 10 pulses = 1 micron.
;

define POSITION command
        long N
  #tyo ( DEVNUM )
  #tyo ( ascii D )
  mvzer ( BUFF , 10 )
  print #i 0 , N , #n
```

```
  #tyo ( CRCHR )
  encode ( BUFF )
;  wrl ( MOTCH , BUFF )
  bfwrl ( BUFF )
;  rdl ( MOTCH , BUFF ) drop
end
; Set scale for number motor pulses per least significant digit
;  of position data. The value of scale factor is integer and
;  varies from 1 to 255 inclusive.
;
; define SCALEFACTOR command
;       integer NUM
;    #tyo ( DEVNUM )
.    #tyo ( ascii U )
,    #tyo ( ascii S )
;    print #i 0 , NUM , #n
;    #tyo ( CRCHR )
;    encode ( BUFF )
;   wrl ( MOTCH , BUFF )
;    rdl ( MOTCH , BUFF ) drop
; end
;
; Read the set scale factor.
;
; define READSCALE
;   MOTCOM ascii U , ascii R
, end




;
; Start the move

define STT command
;  TOTAL += xtnd ( STEPSIZE )
;  if ( ( TOTAL ) UPLIMIT ) or ( TOTAL ( LOLIMIT ) )
;       BEEP print "LIMITS EXCEEDED"
;  else
  MOTCOM ascii G
;  endif
end
;
; Executes normal mode

; define SETMOTOR command
;       integer STEPSIZE
;   if ( cmdcnt ==0 )
;       STEPSIZE := ASTEPSIZE
;   endif
;   BXON
;   mvzer ( BUFF , 10 )
,   NORMALMODE
;   ACCEL 1.0
;   VELOCITY 1.0
,   POSITION xtnd ( STEPSIZE )
; end
;
```

```
; Sets to  the  absolute position desired.
;
; define SETABS
;       integer STEPSIZE
; local
;       long RELSTEP
;            BXON
;            ABSPOS
;            CONVERT
;       RELSTEP :=  LLVAL - XTND ( STEPSIZE )
;  mvzer ( BUFF , 10 )
;             NORMALMODE
;            ACCEL 1.0
;            VELOCITY 1.0
;            POSITION RELSTEP
```

```
(*
 This program is an attempt to use an iterative method to determine
 square roots of real numbers.
*)

(* Calling sequence:
   SQUAREROOT := SQRT ( ARG1 )
*)

real POSRAD
real OLD NEW

define SQRT real
        real RAD
local
     real ACCUR
  POSRAD := FABS ( RAD )
  ACCUR := 0.000001 * POSRAD
  OLD := 0.00
  NEW := 1.00
  while ( FABS ( NEW * NEW - POSRAD ) )= ACCUR )
    OLD := NEW
    NEW := ( OLD + ( POSRAD / OLD ) ) / 2.0
repeat
SQRT := NEW
end
```

```
ext     FZVMOV
ext     VIDAUTO
ext     ERGPOS
ext     FVIDFOCUS
ext FOCUS

mvstr ( "staget" , promstr )

integer STGCBF ( 15. )
integer STPFLAG

define CONNECT_2_MASTER
 STPFLAG off
 INITREC
 begin
        RECEIVE ( STGCBF )
 until ( STPFLAG )
end


define RECONNECT
  SET ( SYNC2 )
  begin
        RECEIVE ( STGCBF )
  until ( STPFLAG )
end




integer TMPICH  TMPOCH

define STOPCO
        integer TERM
  TMPICH := cich
  TMPOCH := coch
  cich := open ( TERM , 'rwa )
  coch := cich
  poke ( 2 , fdb ( coch ) )
  atterm
  STPFLAG on
end

define STRTCO
  detterm
  close ( cich )
  cich := TMPICH
  coch := TMPOCH
  STPFLAG off
  RECONNECT
end

define STGINI
  mvstr ( "TT2:" , TERM )
  INITBOXX
```

```
  mvstr ( "TT3:" , TERM )
   INITMOT
  with ZAXIS
   BXON
   BXON
 end

 define INITSTG
 ;  STGINI
  OFFX := 0L
  OFFY := 0L
  ALPHA := 0.999835
  BETA := 0.018175
  CALFLG OFF
  CONNECT_Z_MASTER
 end

 $RESTART := BASE INITSTG         ; RESTART FOR DEMO PACKAGE

 SAVE WFSTAGE7
```

```
;
;        NAME . Z_MOVE.MG
;
(*       THIS PROGRAM MOVES THE MOTOR IN ANY DIRECTION
         THRO' THE STEPSIZE SPECIFIED IN THE PROGRAM INITZ . *)
;
define Z_MOVE
 STT
end
;
;        NAME : INITZ.MG
;
(*       THIS PROGRAM INITILIZES MOTOR PARAMETERS. THE STEP
         SIZE HAS TO BE SPECIFIED.  (  INITZ ( STEPSIZE )  ) *)
;
define INITZ
WITH ZAXIS
  BXON
 mvzer ( BUFF , 10 )
 NORMALMODE
 ACCEL 1.0
 VELOCITY 1.0
end
;
;        NAME : INITZ 1.MG
;
(*       THIS PROGRAM INITILIZES MOTOR PARAMETERS. THE STEP
```

```
        SIZE HAS TO BE SPECIFIED.  (  INITZ ( STEPSIZE )  ) *)
;
define INT1
 with ZAXIS
  BXON
 mvzer ( BUFF , 10 )
 NORMALMODE
 ACCEL 2.0
 VELOCITY 5.0
end
;
;       NAME : INITZ2.MC
;
(*      THIS PROGRAM INITILIZES MOTOR PARAMETERS. THE STEP
        SIZE HAS TO BE SPECIFIED.  (  INITZ ( STEPSIZE )  ) *)
;
define INT2
 with ZAXIS
 BXON
 mvzer ( BUFF , 10 )
 NORMALMODE
 ACCEL 2.0
 VELOCITY 7.5
end
;
define UPFAST
        long STEP10




  INT1
  STEP10 := ( LABS ( STEP10 ) )
  POSITION   STEP10
        Z_MOVE
end
;
define DNFAST
        long STEP8
  INT2
  POSITION -- ( LABS ( STEP8 ) )
  if ( STEP8 ) 939900L )
        print " ERROR1 "
  else
                Z_MOVE
  endif
 end
;
define UPTST
        INTEGER STEP1 STNUM
  INITZ
  STEP1 := ( ABS ( STEP1 ) )
  POSITION xtnd ( STEP1 )
  iter STNUM
        Z_MOVE
  loop
end
;
```

```
define DNTST
        integer STEP2 STNM
  INITZ
  POSITION xtnd ( -- (  ABS ( STEP2 ) ) )
  iter STNM
        Z_MOVE
  loop
 end


(*      Set the bias, gain, and integration time.
B       SETPARM ( BIAS GAIN IT )
define  SETPARM
        integer B1 G1 I1
                SET.BIAS ( B1 )
                SET.GAIN  ( G1 )
B               SET.IT ( I1 )
                SET.SENS
end

define DARKPARM
     SET.IT ( 255 )
     SET.BIAS ( 0 )
     SET.GAIN ( 10 )
     SET.SENS
;     GAINADJUST
 ;    SET.SENS




end

define BRIGHTPARM
    SET.IT ( 24 )
    SET.BIAS ( 0 )
    SET.GAIN ( 10 )
    SET.SENS
  ;  GAINADJUST
   ; SET.SENS
end *)
```

```
integer MRKT$ ( 0 )
.word   bytewd ( 5 , 23. )
.word   23.
.blkw   1
.word   1
.word   0

integer WTSE$ ( 0 )
.word   bytewd ( 2 , 41. )
.word   23.

define DELAY
        integer DTIM
  MRKT$ ( 2 ) := DTIM * 6
  RSXDIR ( MRKT$ ) ioerr
  RSXDIR ( WTSE$ ) ioerr
end
;
(*
integer FLAG1 STEPS  PR MAXVAL NSTEP PR2 FCFUN PCENT PR1
integer OFF1 LOG1 RFLAG
long    Z0
; PROFUNC := AVERPTR
;
;
;       NAME . AUTOFC




;
define COARSE
        local
                integer TEST FLAG6 CURX CURY FLAG8
        INITZ
        STEPS := 10
        STEPS := ( ABS  ( STEPS ) )
        POSITION xtnd ( STEPS )
        POKE ( 1000k , DBR )
        FLAG1 OFF
        MAXVAL OFF
        CURX OFF
        CURY OFF
        FLAG8 OFF
begin
        FLAG6 OFF
        STT
        RDLN ( OUTLN1 , 128. , 383. , 256. , 256. )
        WBUS?Y
        POKE ( 1000k , DBR )
        CURY := exec ( FCFUN )
        print SLOPE
;        with ZAXIS
;        BXON
;        ABSPOS
;        PRINT CONVERT
;       if ( CURY ) MAXVAL )
```

```
;               MAXVAL := CURY
;       endif
 :       increment CURX
;       PCENT := iscal ( CURY , 100 , MAXVAL )
;       if ( ( PCENT ( PR1 ) and ( FLAG8 ==0 ) )
;               BEEP
;               DNFAST ( 10L )
;               DNFAST ( 100L )
 ;               DELAY ( 1 )
;               UPFAST ( 20L )
;               INITZ
;               STEPS := 10
;               STEPS := ( ABS ( STEPS ) )
;               POSITION xtnd ( STEPS )
 ;               DELAY ( 1 )
;               PR2 := iscal ( MAXVAL 90 100 )
;               MAXVAL OFF
;               FLAG6 ON
;               FLAG8 on
;       endif
        until ( CURY )= PR2 )
end


define VERYFINE



local
        integer PERCENT STOP CURX CURY
 STEPS := 3
 INITZ
 POSITION xtnd ( STEPS )
 POKE ( 1000k , DBR )
 MAXVAL OFF
 STOP OFF
 NSTEP OFF
 CURX OFF
 CURY OFF
 begin
        STT
        RDLN ( OUTLN1 , 128. , 383. , 256. , 256. )
        WBUS?Y
        POKE ( 1000k , DBR )
        CURY := exec ( FCFUN )
        if ( CURY ) MAXVAL )
                MAXVAL := CURY
        endif
        PERCENT := iscal ( CURY , 100 , MAXVAL )
        if ( CURY () MAXVAL )
                increment NSTEP
                if ( PERCENT ( PR )
                        STOP ON
                endif
        endif
```

```
        increment CURX
  until ( STOP  )
  ;      DNTST ( NSTEP  1 )
 end


define AUTOFC
LOCAL
  long ZPOS
        COARSE
        VERYFINE
; WITH ZAXIS
  ;    ABSPOS
   ;   ZPOS := CONVERT
   ; PRINT Z0
 ; if ( RFLAG == 1 )
 ; if ( LABS ( Z0 - ZPOS ) ) 20L )
   ;   SET
    ; POSITION ( Z0 - ZPOS )
     ; STT
; DELAY ( 2 )
; else
  ;  Z0 := ZPOS
, endif
  ; else




    ; Z0 := ZPOS
 ; endif
; PRINT Z0
end

STEPS := 20
MAXVAL OFF
NSTEP OFF
FCFUN := base SLOPE
PR := 180.
PR2 := 1000
PR1 := 75.
OFF1 := 50
*)
```

```
integer BXCH DIROW COL
long OFFX OFFY STX STY STXP STYP BOFFX BOFFY CORRX
long XCOR YCOR WAFXCOR WAFYCOR STXCOR STYCOR CORRY
real ALPHA BETA XPITCH YPITCH
integer FXSIZE FYSIZE OVLX OVLY FXIND FYIND
integer FWX
integer FWY
integer FRX
integer FRY

BOFFX := 0L
BOFFY := 0L

.mac
label $ttys
        mov     r0 , @ # ptr ( tempr0 )
        mov     @ # ptr ( bufptr ) , r0
        movb    (rp)+ , (r0)

        cmpb    # 15k , (r0)+
        bne     0$
        clrb    -(r0)
        mov     # 21. , -(rp)
        mov     # bytewd ( 2 33. ) , -(rp)
        emt     377k
0$:



        mov     @ # ptr ( tempr0 ) , r0
        inc     @ # ptr ( bufptr )
        mov     # bytewd ( 1 115. ) , -(rp)     ; return from ast
        emt     377k
.end

make '$grabs $attach 1410k $ttys 0 base NO_OP ; detterm

define grabs
  apush cich
  cich := bxch
  $grabs
  apop
end

define bwrl
        integer buff
  bufptr := buf
  CLEF$ ( 21. )
  wrl ( bxch buff )
  wtse ( 21. )
end

(* initializes and opens a channel for argolux communication *)
```

```
define INITBOXX
  BXCH := open ( TERM , 'rwa )
  poke ( 2 , fdb ( BXCH ) )
  grabs
end

(* routine to send an ASCII character at a time *)

define BXCOM command
        integer CHAR
  iter cmdcnt
        #tyo ( CHAR )
        nxtarg
  loop
  #tyo ( CRCHR )
  encode ( BUFF )
  bwrl ( BUFF )
end

(* routine to move the stage to the home coordinates *)

define HOME
 BXCOM ASCII H
end

(* routine to move the stage to the desired absolute coordinates




   MOVE ( x-coordinate , y-coordinate : both are long integers ) *)

define MOVE
    long XMOV YMOV
  #tyo ( ASCII M )
  #tyo ( ASCII X )
  mvzer ( BUFF , 10 )
 print #I 0 , XMOV , " , " , #N
  #tyo ( ASCII Y )
 print #I 0 , YMOV , #N
  #tyo ( 15K )
 encode ( BUFF )
 bwrl ( BXCH , BUFF )
end

(* routine to request the current location of the stage wrt the home
   coordinate *)

define POSREQ
  BXCOM ASCII P
  print STR ( BUF )
end

(* splits the transmitted position string into x-coordinate *)

define SPLX long
 local
```

```
    integer  ALX ( 10 )
   #FIELD ( BP ( BUF ) + 2 , 7 , 7 )
     ENCODE ( ALX )
 if ( LLITERAL ( ALX ) )
    SPLX := LLVAL
 endif
 end

 (* splits the transmitted position string into y-coordinate *)

 define SPLY long
   local
      integer ALX ( 10 )
 #FIELD ( BP ( BUF ) + 12 , 7 , 7 )
   ENCODE ( ALX )
 if ( LLITERAL ( ALX ) )
     SPLY := LLVAL
 endif
 end

 (* Attachement to the inspection plan and status data base *)

 define ATIPSDB
   WITH M_IPSDB
   ATTRG ( "IPSDBR" , 160000K )
   ptr ( IPSDB_REC ) := WNDADR
   with INSP_PLN




   with INSP_DATA_BASE
 end

 (* transforms the stage coordinates into the wafer coordinate system *)


 define STWAFTR
          real POSX POSY
 WAFXCOR := LFIX ( ALPHA * POSX  + BETA * POSY  )
 WAFYCOR := LFIX ( -- BETA * POSX + ALPHA * POSY )
 end

 (* transforms the wafer coordinates into the stage coordinates *)

 define WAFSTTR
          real POSX POSY
 STXCOR := LFIX ( ALPHA * POSX  + ( -- BETA * POSY ) )
 STYCOR := LFIX ( BETA * POSX + ALPHA * POSY )
 end

 (* computes the x-coordinate of the desired die *)

 define DIEX REAL
      DIEX := ( FLOAT ( DIROW ) * XPITCH )
 end

 (* computes the y-coordinate of the desired die *)
```

```
define DIEY REAL
        DIEY := ( FLOAT ( COL ) * YPITCH )
end

(* computes the x-coordinate of the desired feature *)

define FWAX integer
    FWAX := FWX
end

(* computes the y-coordinate of the desired feature *)

define FWAY integer
     FWAY := FWY
end

(* computes the x-coordinate of the desired frame *)

define FRAX integer
local
      integer C
     C := FRX
FRAX := C + FXIND * ( (  FXSIZE * ( 100 - OVLX ) ) / 100 )
end

(* computes the y-coordinate of the desired frame *)




define FRAY integer
   local
    integer C
    C := FRY
FRAY := C + FYIND * ( ( FYSIZE * ( 100 - OVLY ) ) / 100 )
end

(* computes the x-coordinate of the desired frame in a given feature in a
   given die *)

define FRAMX
   local
    real ALX
    integer APX AXX
     ALX := DIEX
     APX := FWAX
     AXX := FRAX
XCOR := xtnd ( APX + AXX ) + LFIX ( ALX )
end

(* computes the y-coordinate  of the desired frame in a given feature
   in a given die *)

define FRAMY
   local
   real ALX
```

```
    integer APX APY
    ALX := DIEY
    APX := FWAY
    APY .= FRAY
 YCOR := xtnd ( APX + APY ) + LFIX ( ALX )
 end

 (* defines a backlash correction on the coordinates depending
    on the direction of move *)

         STXP := OFFX
         STYP := OFFY

 define BLCORR
  local
      integer LDIR LDIR1 LDIR2 LDIR3
  LDIR1 ON
  LDIR2 ON
 if ( STX )= STXP )
    LDIR off
 else
    LDIR on
 endif
   if ( LDIR <> LDIR1 )
    LDIR1 := LDIR
   if ( STX < 0L )
    STX := STX - BOFFX

   else
    STX := STX + BOFFX
   endif
  endif
 if ( STY )= STYP )
    LDIR3 off
 else
    LDIR3 on
 endif
  if ( LDIR2 <> LDIR3 )
    LDIR2 := LDIR3
  if ( STY < 0L )
  STY := STY - BOFFY
 else
  STY := STY + BOFFY
 endif
 endif
 STXP := STX
 STYP := STY
 end

 (* moves the stage to the desired frame *)

 define STFRAM
  local
    long POSX POSY
    real XMOV YMOV
```

```
with INSP_DATA_BASE
STAGE_ERR := TRUE
        FRAMX
        XMOV := LFLOAT ( XCOR )
        FRAMY
        YMOV := LFLOAT ( YCOR )
        WAFSTTR ( XMOV , YMOV )
STX := -- STXCOR + OFFX + CORRX
STY := -- STYCOR + OFFY + CORRY
print STX , STY , STXCOR , STYCOR
        BLCORR
MOVE ( STX , STY )
; POSREQ
; POSX := SPLX
; POSY := SPLY
; if ( ( POSX <> STX ) AND ( POSY <> STY ) )
; STAGE_ERR := FALSE
; endif
end

(* The stage moving function to be called from the Master *)

define STAGEM
local
     integer SITE FRAM
 ATIPSDB
 STAGE_BUSY := TRUE




 SITE := DES_SITE
 FRAM := DES_FRAME
 with DES_RET
     DIROW := ROW
     COL := CLMN
 print DIROW , COL
 with INSP_PLN
 with HEADER
       XPITCH := DIE_X
       YPITCH := DIE_Y
 with LAYERS ( DES_LAYER )
 with DTL_LAYER_REV ( #_REVS - 1 )
 with L_RETICLE
 with RETICLE_DIE
 with D_PATTERNS ( DES_PATTERN )
 with S_ORG ( SITE )
    FWX := X
    FWY := Y
with F_ORG
     FRX := X
     FRY := Y
 with F_DESCR
 with F_SZ
      FXSIZE := X
      FYSIZE := Y
 with F_OLAP
      OVLX := X
```

```
      OVLY := Y
 with INSP_FR ( SITE )
 with FRAMES ( FRAM )
      FXIND := ROW
      FYIND := CLMN
               STFRAM
  STAGE_BUSY := FALSE
 with CUR_RET
      ROW := DIROW
      CLMN := COL
      CUR_SITE := DES_SITE
      CUR_FRAME := DES_FRAME
DREGION
end
(* defines a correction to the coordinates due to the error in positioning *)

define ERRCORR
local
long POSX1 POSY1 POSX2 POSY2
        CORRX := 0L
        CORRY := 0L
        STAGEM
        POSREQ
        POSX1 := SPLX
        POSY1 := SPLY
print "Manually align the feature"
        pause




        POSREQ
        POSX2 := SPLX
        POSY2 := SPLY
        CORRX := POSX2 - POSX1
        CORRY := POSY2 - POSY1
end

(* STAGE CALIBRATION  procedure *)

integer CALFLG
        CALFLG on

define INITOFF
 ATIPSDB
 print "Manually define home, press 'DEFINE HOME' button"
 print "Manually position origin of die (0,0)"
 print "Press COMP ON at the stage control panel"
 pause
 POSREQ
 OFFX := SPLX
 OFFY := SPLY
end


define CALSTG
    local
```

```
      long POSX1 POSY1
      long D D1
 if ( CALFLG )
 INITOFF
 print "Move to last die in X direction"
 print "Press COMP ON at stage control panel"
 pause
    POSREQ
    POSX1 := SPLX
    POSY1 := SPLY
D := POSX1 - OFFX
D1 := POSY1 - OFFY
ALPHA := LFLOAT ( -- D ) / SQRT ( LFLOAT ( D ) * LFLOAT ( D ) + ^
                     LFLOAT ( D1 ) * LFLOAT ( D1 ) )
BETA := LFLOAT ( -- D1 ) / SQRT ( LFLOAT ( D ) * LFLOAT ( D ) + ^
                         LFLOAT ( D1 ) * LFLOAT ( D1 ) )
print #F 12 6 , "alpha: " , ALPHA , "beta: " , BETA
print OFFX , OFFY
endif
end

(* Switch the illumination according to the current and desired set in
IPSDB *)

define ILLSW
 ATIPSDB
 if ( CUR_ILLUM () DES_ILLUM )




  CUR_ILLUM := DES_ILLUM
  DREGION
  ATIOPAGE
  POKE ( 200K , CSR )     ; switch the illumination
  DELAY ( 3 )
   POKE ( 0 , CSR )       ; make it stable.
                          ; Note: no handshaking with the hardware
  DELAY ( 60 )            ; Wait to make sure switch happens before we return
 endif
DREGION
end

(* Switch the magnification according to the current and desired set in
IPSDB *)

define LENSW
 ATIPSDB
 if ( CUR_MAGNF () DES_MAGNF )
  CUR_MAGNF := DES_MAGNF
  DREGION
  ATIOPAGE
  POKE ( 400K , CSR )     ; switch the magnification
  DELAY ( 1 )
  POKE ( 0 , CSR )        ; make it stable.
                          ; Note: no handshaking with the hardware
 endif
```

```
DREGION
end
```

```
integer SLPOFF SLPSCL
.MAC
entry SLOPE integer
        mov     # 128. , r0                     ; 256 points, index
        mov     @ # ptr ( SLPOFF ) , r1
        asl     r1
        sub     r1 , r0
        mov     # ptr ( OUTLN ) , -(msp)        ; store pointer to the array
        add     r0 , (msp)                      ; point to last elt in array
        clr     r3                              ; clear maximum
0$:     add     @ # ptr ( SLPOFF ) , (msp)
        movb    @ 0 (msp) , r2                  ; get OUTLN1 ( r0 + 1 )
        bic     # 177400k , r2                  ; clear high byte from movb
        dec     (msp)
        sub     @ # ptr ( SLPOFF ) , (msp)      ; decrement pointer
        movb    @ 0 (msp) , r1                  ; get OUTLN1 ( r0 - 1 )
        bic     # 177400k , r1                  ; clear high byte
        sub     r1 , r2                         ; get slope
        tst     r2
        bpl     1$                              ; see if negative
        neg     r2                              ; yes , make positive
1$:     cmp     r3 , r2                         ; see if > than current max
        bge     2$
        mov     r2 , r3                         ; yes, store new max
2$:     dec     r0
        bne     0$
```

```
        mov     r3 , (msp)                      ; return maximum slope
        next
 .end


(*
DEFINE PEAK INTEGER
 LOCAL
        INTEGER TEMP1
 PEAK OFF
 ITER 256.
        MVBYWD ( OUTLN1 , I , OUTLN , 64 )
        TEMP1 := OUTLN ( VMAX ( OUTLN , 64 ) ) - OUTLN ( VMIN ( OUTLN , 64 ) )
        PEAK := PEAK + TEMP1
 LOOP ( 64. )
 PEAK := URSHIFT ( PEAK , 1 )
END   *)


SLPSCL := 1.
SLPOFF := 1.
```

COMPARISON BETWEEN REFERENCE AND EDGES

```
(* ******************************************************************
        MATCHT.MG - THIS MODULE LOADS ALL OF THE MODULES USED IN "MATCHT"
****************************************************************** *)

ext     PDPID                    ; Assembly language mneumonics.
ext     DMISC                    ; Miscellaneous utility routines.
ext     MCHREG                   ; Region mapping utilities for MATCHT.
ext     MCHCOM                   ; Intertask communication utilities for MATCHT.
ext     [5,1]INSPLAN             ; Inspection Data Base record structure
ext     22BADDR                  ; 22 bit address suppport.
ext     GRABIM                   ; Frame grabber support.
ext     MODELT                   ; Model record structure.
ext     OPLINE                   ; Routine to 'open' a line between two points.
ext     MDLMTCH                  ; Matching and Image Registration routines.

mvstr ( "matcht" , promstr )

integer MCHCBF ( 15. )           ; Intertask communication buffer.

integer STPFLAG                  ; Flag to indicate no communication.
integer TMPICH  TMPOCH           ; Temporaries for input and output channels.

(*      Start communication with the master task              *)
define CONNECT_2_MASTER
 INITREC




 begin
        RECEIVE ( MCHCBF )
 until ( STPFLAG )
end


(*      Restart communication with the master task.           *)
define RECONNECT
  SET ( SYNC2 )
  begin
        RECEIVE ( MCHCBF )
  until ( STPFLAG )
end


(*      Stop communication with master task and allow input from a terminal.
        STOPCO ( 'TTn )                  *)
define STOPCO
        integer TERM
  TMPICH := cich
  TMPOCH := coch
  cich := open ( TERM , 'rwa )
  coch := cich
  poke ( 2 , fdb ( cich ) )
  atterm
  STPFLAG on
end
```

```
(*      Restart communication after a STOPCO has been executed.          *)
define STRTCO
  detterm
  close ( cich )
  cich := TMPICH
  coch := TMPOCH
  STPFLAG off
  RECONNECT
end

(*      Initialisation routine for MATCHT.                  *)
define MATCHINIT
;  with M_MODEL
;    ATTRG ( "MODELR" , 160000k )
;  with M_EDGE
;    ATTRG ( "EDGIMG" , 140000k )
  CONNECT_2_MASTER
end

$restart := base MATCHINIT

save WFMATCHT




parameter WCR := 172410k         ; DMA word count register.
parameter BAR := 172412k         ; Bus address register for DMA.
parameter CSR := 172414k         ; Control status register.
parameter DBR := 172416k         ; Data buffer register.

long    PHYADR                   ; Physical (22-bit) address of the buffer.

make    'MRKT$  rsxcall bytewd ( 5 23. )

(*      Delay function using Mark Time directive.  DELAY_TIME is the number
        of seconds times 10.  e.g.  DELAY_TIME = 10. is a delay of 1 second.
        DELAY ( DELAY_TIME )                    *)
define DELAY
        integer DEL
  MRKT$ ( 23. , DEL * 6 , 1 , 0 )
  WAIT ( 23. )
end

(*      Wait until DMA operation is complete.  (Monitors BUSY bit.)      *)
define WBUS?Y
  while ( peek ( CSR ) AND 200k )
  repeat
end
```

```
(*      Read a line from the frame grabber.
        RDFGLN ( BUFFER , X0 , XL , Y0 , YL )              *)
define RDFGLN
        integer BUFF ( 1 ) X0 XL Y0 YL
  PHYADR := 22ADDR ( BUFF )
  poke ( 130000k + X0 - 1 , DBR )
  poke ( 114000k + Y0 , DBR )
  poke ( -- XL / 2 , WCR )
;  poke ( -- ( XL * YL ) , WCR )
  poke ( lsword ( PHYADR ) , BAR )
  poke ( 0 , DBR )
  poke ( msword ( PHYADR ) + 1 , CSR )
end


(*      Get an image from the frame grabber into a memory region.        *)
define GETIM
 local
        integer BUFPTR
 with M_MODEL
  ATTRG ( "IOPAGE" , 160000k )
  poke ( 1000k , DBR )
  DELAY ( 1 )
  poke ( 0 , DBR )
 with M_EDGE
  ATTRG ( "EDGIMG" , 140000k )




  WNDOFF off
  MAPW ( WNDB ) ioerr
  iter 256.
    REMAP ( i ) drop
    BUFPTR := WNDADR
      do 128. 159.
        WBUS?Y
        RDFGLN ( BUFPTR , 128. , 256. , i + j , 1 )
        BUFPTR += 256.
      loop
  loop ( 32. )
  WBUS?Y
  DREGION
 with M_MODEL
  DREGION
end


define COPYIM
 local
        integer EDGADR
 with M_MODEL
  ATTRG ( "MTCHIM" , 160000k )
  WNDOFF off
  MAPW ( WNDB ) ioerr
 with M_EDGE
  ATTRG ( "EDGIMG" , 140000k )
```

```
    EDCADR := WNDADR
    WNDOFF off
    MAPW ( WNDB ) ioerr
    iter 256.
      with M_EDGE
           REMAP ( i ) drop
      with M_MODEL
           REMAP ( i ) drop
           mvwds ( EDCADR , WNDADR , 4096. )
    loop ( 32. )
    DREGION
  with M_EDGE
    DREGION
end
```

```
(*                MODEL RECORD

          Type of point           Code
          -------------           ----

          Right Angle Corner      4!X!Y
          Endpoint                0!X

          where X is the orientation of the point and Y is the direction
          as follows:

          *-------        0                 -------*         2

              *                                !
              !                                !
              !           1                    !             3
              !                                !
              !                                *


          *------            ------*           !       !
          !                        !           !       !
          !   4                5   !         6 !       !  7
          !                        !     ------*       *-------
```

```
*)

parameter MAX_#_ENT := 20        ; Maximum # of permissible entities.
parameter #POINTS := 25          ; Maximum # of points permitted within
                                 ;  an entity.


record POINT_REC
        integer XI YI            ; Coordinates of first corner point.
        integer CURTYPE          ; Type of the line between 1st and 2nd point.
        integer XJ YJ            ; Coordinates of second point.
        integer NXTTYPE          ; Type of the next line (between 2nd and 3rd).
        dummy -3
endrecord

record ENTITY
        integer #PTS                      ; # of points.
        POINT_REC ZI ( #POINTS )          ; See record POINTS.
endrecord


record FRAME_REC
        integer FRM#             ; Frame # .
        integer #ENT             ; # of entities.
        ENTITY EI ( MAX_#_ENT )
        integer HX1     HX2     HY1     HY2     ; Horizontal landmark points.




        integer VX1     VX2     VY1     VY2     ; Vertical landmark points.
        integer HX3     HX4     HY3     HY4     ; Horizontal landmark points.
        integer VX3     VX4     VY3     VY4     ; Vertical landmark points.
endrecord


(*      Store the current model under a given filename.
        STORE_MODEL ( 'FILENAME )
define STORE_MODEL
        integer NAME
 local
        integer OUTCH
  OUTCH := open ( NAME , 'wt )
  wrs ( OUTCH , FRAME , size FRAME_REC )
  close ( OUTCH )
end
*)


(*      Load the current model from a previously stored file.
        GET_MODEL ( 'FILENAME )                    *)
define GET_MODEL
        integer NAME
 local
        integer INCH
        char    PNAME ( 30. )
 with M_MODEL
```

```
   ATTRG ( "MODELR" , 160000k )
   ptr ( FRAME_REC ) := WNDADR
   mvstr ( 'dm4:[5,3] , PNAME )
   concat ( PNAME , NAME )
   INCH := open ( PNAME , 'r )
   rds ( INCH , WNDADR , size FRAME_REC ) drop
   close ( INCH )
   DREGION
 end
```

```
long    CURPNT                    ; Coordinates of the current point.

.mac

integer LNTMP ( 0 )
.blkw   8.
.WORD   ptr ( LNTMP ( 3 ) )


; Line Temporaries:
;  LNTMP ( 0 ) := NPTS           ; Major Direction Length (Count).
;  LNTMP ( 1 ) := DMN            ; Change in minor direction.
;  LNTMP ( 2 ) := DMJ            ; Change in major direction.
;  LNTMP ( 3 ) := YINC           ; Y increment if major, else 0.
;  LNTMP ( 4 ) := XINC           ; X increment if major, else 0.
;  LNTMP ( 5 ) := YINC           ; Y increment.
;  LNTMP ( 6 ) := XINC           ; X increment.
;  LNTMP ( 7 ) := FRAC           ; accumulated fraction
;  LNTMP ( 8 ) := ptr ( XINC )   ; Pointer to current increments.
;


(*      Set up the endpoints of the line and initialize the current
        point (CURPNT).  Initialize the LNTMP array.
        FIRSTPT ( X1 Y1 X2 Y2 )                *)
entry FIRSTPT
```

```
        mov     # LNTMP , r0            ; Get pointer to LNTMP array.
        mov     (msp)+ , r3             ; Get Y1.
        mov     (msp)+ , r2             ; Get X1.
        sub     (msp) , r3              ; Form Y1 - Y0.
        bge     0$                      ; If negative:
        neg     r3                      ;  make it positive and
        mov     # -1 , r1               ;  form -1 as increment.
        br      1$                      ; Else:
0$:     mov     # 1 , r1                ;  form +1 as increment.
1$:     mov     r1 , 6 (r0)             ; Store increment in LNTMP ( 3 )
        mov     r1 , 10. (r0)           ;  and LNTMP ( 5 ).
        mov     (msp)+ , @ # ptr ( CURPNT )         ; Store in CURY.

        sub     (msp) , r2              ; Form X1 - X0.
        bge     2$                      ; If negative:
        neg     r2                      ;  make it positive and
        mov     # -1 , r1               ;  form -1 as increment.
        br      3$                      ; Else:
2$:     mov     # 1 , r1                ;  form +1 as increment.
3$:     mov     r1 , 8. (r0)            ; Store increment in LNTMP ( 4 )
        mov     r1 , 12. (r0)           ;  and LNTMP ( 6 ).
        mov     (msp)+ , @ # ptr ( CURPNT ) + 2     ; Store in CURX.

        cmp     r2 , r3                 ; Compare DX to DY.
        blt     4$                      ; If DX ) DY:
        mov     r3 , 2 (r0)             ;  Place DY in LNTMP ( 1 ).
        mov     r2 , r3                 ;  Copy DX into r3.




        clr     6 (r0)                  ;  Don't change Y on major scale.
        br      5$                      ; Else:
4$:     mov     r2 , 2 (r0)             ;  Place DX in LNTMP ( 1 ).
        clr     8. (r0)                 ;  Don't change X on major scale.

5$:     mov     r3 , 4 (r0)             ; Place major in LNTMP ( 2 )
        mov     r3 , (r0)               ; Initialize counter -- LNTMP ( 0 ).
        inc     (r0)                    ; Include the endpoints.
        neg     r3                      ; Form -- ( MAJOR / 2 )
        asr     r3
        mov     r3 , 14. (r0)           ; Place in LNTMP ( 7 ) (fraction).
        next

.local

(*      Update the contents of CURPNT to point to the next point in the line.
        If there is a next point, TRUE is returned, else FALSE is returned.
        LOGICAL := NXTPNT                               *)
entry NXTPNT integer
        mov     # LNTMP , r0            ; Get pointer to LNTMP array.
        dec     (r0)                    ; Decrement counter.
        bne     0$                      ; If no more points,
        clr     -(msp)                  ;  push a zero (false)
        br      2$                      ;  and return.

0$:     mov     # ptr ( CURPNT ) , r2   ; Get pointer to CURPNT.
```

```
        mov     16. (r0) , r1          ; Get pointer to XINC.
        add     2 (r0) , 14. (r0)      ; Add minor change to fraction.
        blt     1$                     ; If overflow,
        sub     4 (r0) , 14. (r0)      ;  Subtract major change from fraction.
        add     # 4 , r1               ;  Point to major and minor increments.
1$:     add     (r1)+ , (r2)+          ; Add proper X increment to CURX.
        add     (r1) , (r2)            ; Add proper Y increment to CURY.
        mov     # -1 , -(msp)          ; Push a -1 (true)
2$:     next                           ;  and return.

.end

(*      Update the contents of CURPNT after moving a given number of points.
        -1 is returned if successful.  If not successful, the number of
        successful increments before ending is returned.
        IVAL := NXTPNTS ( #_POINTS )                            *)
define NXTPNTS integer
        integer #PNTS
  NXTPNTS on
  iter #PNTS
        if ( not NXTPNT )
                NXTPNTS := i
                exit
        endif
  loop


end
```

```
integer CORLEN  CORWID                    ; Length and width of corner masking.
long    PNTINC                            ; Point increment to add to CURPNT.
integer DX1     DY1     DX2     DY2       ; Deltas for best fit of line to image.
integer MDLEV                             ; Grey level to write for model points.
integer TOTAL   BEST                      ; Totals for finding best fits.
integer REGX    REGY                      ; Registration error in X and Y.
integer XWIND   YWIND                     ; Registration window size in X and Y.

XWIND := 4
YWIND := 4

long    VMASK ( 0 )                       ; Masking values for vertical lines.
.blkw   8
.long   wdlong ( 0 , 1 )
.long   wdlong ( 0 , -1 )
.long   wdlong ( 0 , -1 )
.long   wdlong ( 0 , 1 )

long    HMASK ( 0 )                       ; Masking values for horizontal lines.
.blkw   8
.long   wdlong ( 1 , 0 )
.long   wdlong ( 1 , 0 )
.long   wdlong ( -1 , 0 )
.long   wdlong ( -1 , 0 )




MDLEV := 252.               ; Make these points look like evens.
CORLEN := 5
CORWID := 3

.mac

(*      Add two longs as if they were two sets of two integers.
        LVAL := LI+ ( LVAL1 , LVAL2 )                         *)
entry LI+ long
        add     (msp)+ , 2 (msp)
        add     (msp)+ , 2 (msp)
        next

(*      Subtract two longs as if they were two sets of two integers.
        LVAL := LI- ( LVAL1 , LVAL2 )                         *)
entry LI- long
        sub     (msp)+ , 2 (msp)
        sub     (msp)+ , 2 (msp)
        next

.end

(*      Set the size of the registration window.
        WINDOW ( XSIZE , YSIZE )                  *)
define WINDOW
```

```
        integer XW YW
  XWIND := XW
  YWIND := YW
 end


 (*     Test the current line to be vertical.
        LOGICAL := VERTICAL                          *)
 define VERTICAL ifunc
   ( abs ( YJ - YI ) ) abs ( XJ - XI ) )
 end


 (*     Fix the model by adding in the registration error.
        FIXMDL ( REG_X , REG_Y )                     *)
 define FIXMDL
        integer REGX    REGY
   iter #ENT
     with EI ( i )
        iter #PTS
          with ZI ( i )
               XI += REGX
               YI += REGY
        loop
   loop
 end


 (*     Test a model line for matches against the image.  Place total number
        of matches in TOTAL.
        TESTLN ( X1 Y1 X2 Y2 #_SKIPS )                *)
 define TESTLN
        integer X1      Y1      #PTS
        long    REGINC
   if ( #PTS )
     CURPNT := wdlong ( Y1 , X1 )
     iter #PTS + 1
        if ( MRDPIX ( CURPNT ) )
               increment TOTAL
        endif
        CURPNT := LI+ ( CURPNT , REGINC )
     loop
   endif
 end


 (*     Return the number of points to skip for a model line.
        IVAL := GETPTS ( POINT_1 , POINT_2 )          *)
 define GETPTS integer
        integer PT1 PT2
   if ( PT2 ) PT1 )
        GETPTS := limit ( ( PT2 - PT1 ) / 10 , 1 , 10 )
   else
```

```
        GETPTS := limit ( ( PT2 - PT1 ) / 10 , -10 , -1 )
    endif
  end

  (*      Register the image with the model.  Fix the model when registration
          is complete.                                   *)
  define REGISTER
   local
          integer H1#PTS  V1#PTS
          integer H2#PTS  V2#PTS
          long    H1PTINC V1PTINC
          long    H2PTINC V2PTINC
   with M_MODEL
    ATTRC ( "MODELR" , 160000k )
   with M_EDGE
    ATTRC ( "MTCHIM" , 140000k )
    BEST off
    REGX off
    REGY off
    H1#PTS := GETPTS ( HX1 , HX2 )
    H2#PTS := GETPTS ( HX3 , HX4 )
    V1#PTS := GETPTS ( VY1 , VY2 )
    V2#PTS := GETPTS ( VY3 , VY4 )
    H1PTINC := wdlong ( 0 , H1#PTS )
    H2PTINC := wdlong ( 0 , H2#PTS )
    V1PTINC := wdlong ( V1#PTS , 0 )
    V2PTINC := wdlong ( V2#PTS , 0 )




    H1#PTS := ( HX2 - HX1 ) / H1#PTS
    H2#PTS := ( HX4 - HX3 ) / H2#PTS
    V1#PTS := ( VY2 - VY1 ) / V1#PTS
    V2#PTS := ( VY4 - VY3 ) / V2#PTS
    do -- XWIND , XWIND
     do -- YWIND , YWIND
         TOTAL off
         TESTLN ( HX1 + j , HY1 + i , H1#PTS , H1PTINC )
         TESTLN ( HX3 + j , HY3 + i , H2#PTS , H2PTINC )
         TESTLN ( VX1 + j , VY1 + i , V1#PTS , V1PTINC )
         TESTLN ( VX3 + j , VY3 + i , V2#PTS , V2PTINC )
         if ( TOTAL ) BEST )
                BEST := TOTAL
                REGX := j
                REGY := i
         endif
     loop
    loop
    FIXMDL ( REGX , REGY )
  print REGX , REGY
    DREGION
   with M_MODEL
    DREGION
    ATTRC ( "IPSDBR" , WNDADR )
    ptr ( IPSDB_REC ) := WNDADR
    with INSP_DATA_BASE
          REG_X := REGX
```

```
            REG_Y := REGY
      DREGION
    end


    (*      Get the best match of a model line to an image line.  Place results
            in the variables DX1, DY1, DX2, and DY2.
            GETBEST ( X1 , Y1 , X2 , Y2 )                     *)
    define GETBEST
            integer X1      Y1      X2      Y2
     local
            integer #SKIPS
      #SKIPS := limit ( max ( abs ( X2 - X1 ) , abs ( Y2 - Y1 ) ) / 10 , 1 , 10 )
      BEST off
      do -1 1
            TOTAL off
            if ( VERTICAL )
                    FIRSTPT ( X1 + i , Y1 , X2 + i , Y2 )
            else
                    FIRSTPT ( X1 , Y1 + i , X2 , Y2 + i )
            endif
            begin
                    if ( MRDPIX ( CURPNT ) ) increment TOTAL endif
            until ( NXTPNTS ( #SKIPS ) <> -1 )
            if ( TOTAL > BEST )
                    BEST := TOTAL
                    if ( VERTICAL )
                            DX1 := i
                            DX2 := i
                            DY1 off
                            DY2 off
                    else
                            DX1 off
                            DX2 off
                            DY1 := i
                            DY2 := i
                    endif
            endif
      loop
    end


    (*      Look for pixels adjacent to the line to match to.  If there are
            adjacent pixels, delete them from the image.  NOADJ returns TRUE
            if no adjacent pixels were found; otherwise, it returns FALSE.
            LOGICAL := NOADJ                        *)
    define NOADJ integer
      NOADJ on
      if ( MRDPIX ( LI+ ( CURPNT , PNTINC ) ) )      ; >= TESTVAL )
            MWRPIX ( LI+ ( CURPNT , PNTINC ) , 0 )
            NOADJ off
      endif
      if ( MRDPIX ( LI- ( CURPNT , PNTINC ) ) )      ; >= TESTVAL )
```

```
        MWRPIX ( LI- ( CURPNT , PNTINC ) , 0 )
        NOADJ off
  endif
end


(*      Mask the outside of a corner.  The area masked is CORWID x CORWID.  *)
define OUTERMASK
 local
        long    FIRSTPT TMPPNT
  FIRSTPT := LI- ( CURPNT , VMASK ( CURTYPE ) )
  iter CORWID
    FIRSTPT := LI- ( FIRSTPT , HMASK ( CURTYPE ) )
    TMPPNT := FIRSTPT
    iter CORWID
        MWRPIX ( TMPPNT , 0 )
        TMPPNT := LI- ( TMPPNT , VMASK ( CURTYPE ) )
    loop
  loop
end


(*      Mask a corner using the increment supplied.
        CORMASK ( PNTINC )                               *)
define CORMASK
        long    CURINC
 local




        long    TMPPNT
  TMPPNT := CURPNT
  iter CORWID + 1
        MWRPIX ( TMPPNT , 0 )
        TMPPNT := LI+ ( TMPPNT , CURINC )
  loop
  TMPPNT := CURPNT
  iter CORWID
        TMPPNT := LI- ( TMPPNT , CURINC )
        MWRPIX ( TMPPNT , 0 )
  loop
end


(*      Match a model line to the image.  Also perform corner masking.
        MATCHLN ( X1 , Y1 , X2 , Y2 )                    *)
define MATCHLN
        integer X1 Y1 X2 Y2
 local
        long    TMPPNT
        long    NXTINC
 FIRSTPT ( X1 Y1 X2 Y2 )
 if ( VERTICAL )
        PNTINC := VMASK ( CURTYPE )
        NXTINC := VMASK ( NXTTYPE )
 else
        PNTINC := HMASK ( CURTYPE )
```

```
          NXTINC := HMASK ( NXTTYPE )
     endif
     OUTERMASK
     iter CORLEN
          CORMASK ( PNTINC )
          if ( not NXTPNT ) return endif
     loop
     if ( LNTMP ( 0 ) ) CORLEN )
       begin
          if ( MRDPIX ( CURPNT ) )          ; )= TESTVAL )
                 MWRPIX ( CURPNT , 0 )
          else
                 if ( NOADJ ) MWRPIX ( CURPNT , MDLEV ) endif
          endif
          NXTPNT drop
       until ( LNTMP ( 0 ) == CORLEN )
     endif
     begin
          CORMASK ( NXTINC )
     until ( not NXTPNT )
   end


   (*      Execute the matching process for every line in the model.      *)
   define MATCH
    with M_MODEL




     ATTRG ( "MODELR" , 160000k )
    with M_EDGE
     ATTRG ( "MTCHIM" , 140000k )
     iter #ENT
       with EI ( i )
          iter #PTS - 1
            with ZI ( i )
                 GETBEST ( XI YI XJ YJ )
                 MATCHLN ( XI + DX1 , YI + DY1 , XJ + DX2 , YJ + DY2 )
          loop
     loop
     DREGION
    with M_MODEL
     DREGION
   end
```

SUBSTITUTE SHEET

DEFECT ANALYSIS

```
(* **********************************************************************
        DEFECT.MG - THIS MODULE LOADS ALL OF THE MODULES USED IN "DEFECT"
 ********************************************************************** *)

ext     PDPID                  ; Assembly language mneuemonics.
ext     MIXLIB                 ; Mixed-mode arithmetic support.
ext     DMISC                  ; Miscellaneous utilities.
ext     DEFREG                 ; Region mapping utilities.
ext     DEFCOM                 ; Intertask communication utilities.
ext     POINTS                 ; Analysis of disagreement pixels.
ext     [5.1]INSPLAN           ; Inspection Data Base record structure.
ext     CONFIRM                ; Store defects or confirm repeating defects.

integer DEFCBF ( 15. )         ; DEFECT intertask communication buffer
integer STPFLAG                ; Flag to indicate stopped communication.
integer TMPICH  TMPOCH         ; Temporaries for input and output channels.

(*      Start communication with the master task.                *)
define CONNECT_2_MASTER
 INITREC
 begin
        RECEIVE ( DEFCBF )
 until ( STPFLAG )
end


(*      Restart communication with the master task.              *)
define RECONNECT
  SET ( SYNC2 )
  begin
        RECEIVE ( DEFCBF )
  until ( STPFLAG )
end


(*      Stop communication, and allow input from a terminal.
        STOPCO ( 'TTn )                              *)
define STOPCO
        integer TERM
  TMPICH := cich
  TMPOCH := coch
  cich := open ( TERM , 'rwa )
  coch := cich
  poke ( 2 , fdb ( cich ) )
  atterm
  STPFLAG on
end
```

```
(*      Restart communication after a STOPCO.          *)
define STRTCO
  DETTERM
  close ( cich )
  cich := TMPICH
  coch := TMPOCH
  STPFLAG off
  RECONNECT
end


(*      Perform both disagreement analysis and defect storage/confirmation.  *)
define DETECT
  BLOB
  STORE
end


(*      Initialization for DEFECT.       *)
define DEFECTINIT
  CONNECT_2_MASTER
end

mvstr ( 'defect , promstr )

$restart := base DEFECTINIT




save WFDEFECT
```

```
integer BANDSIZE                ; Size of "don't care" band around image.
integer #BLOB                   ; Allowable number of blobs.
#BLOB := 150.
BANDSIZE := 25.

integer X0 XL Y0 YL             ; Image limits.
XL := 256.
YL := 256.

record RASPT_REC
        integer PTCNT                   ; Number of points found.
        integer POINTS ( 256. )         ; Point positions.
endrecord

RASPT_REC RASPTS
with RASPTS

record BLOB_REC
        integer BLBID           ; ID of this blob.
        integer BLBTOT          ; Total points.
        long    XBCOM           ; X center of mass totals.
        long    YBCOM           ; Y center of mass totals.
        integer MINX0   MAXX1   ; Minimum and Maximum X values.
        integer MINY0   MAXY1   ; Minimum and Maximum Y values.
endrecord




record FINAL_REC
        integer CURLINE         ; Y line number.
        integer CURBLOB         ; ID of blob that new point is added to.
        integer CURPT           ; X position of current point.
        integer NXTBLB          ; Pointer to the free blob stack ID.
        integer FREEBLOBS ( #BLOB )
        integer BLOBMAP ( ( #BLOB + 15 ) / 16 )
        BLOB_REC BLOBS ( #BLOB )
endrecord

FINAL_REC FINAL
with FINAL


(*      Record for accessing the image lines with the mapped region.    *)
record LINES_REC
        char    TOP ( 256. )
        char    CEN ( 256. )
        char    BOT ( 256. )
endrecord

.mac

(*      Convert a raster line into point encoded data.
        RAS2PT ( INPUT_BUFFER , OUTPUT_BUFFER , LENGTH )        *)
entry RAS2PT
```

```
        mov     (msp)+ , r0      ; Count -> r0.
        mov     (msp)+ , r1      ; Output buffer pointer -> r1.
        mov     (msp) , r2       ; Input buffer pointer -> r2.
        clr     (msp)            ; Clear temporary point counter.
        mov     r1 , -(rp)       ; Store output pointer on rp stack.
        clr     r3               ; Clear pixel counter.
        tst     (r1)+            , Bump pointer to output buffer.
0$:     tstb    (r2)+            ; Test the point.
        beq     1$               ; If it is not zero:
;       bitb    # 1 , -(r2)      ; Low order bit is 0 if even, 1 if odd.
;       bne     2$               ; If even:
        mov     r3 , (r1)+       ;  Copy pixel number into POINTS array.
        inc     (msp)            ;  Increment point counter.
; 2$:   clrb    (r2)+            ; Zero the image point.
1$:     inc     r3               ; Increment pixel counter.
        dec     r0               ; Decrement count.
        bne     0$               ; Loop back.
        mov     (rp)+ , r1       ; Restore output buffer pointer.
        mov     (msp)+ , (r1)    ; Store point count in PTCNT.
        next

(*      Set all points that are at VAL2 to VAL1 in the vector.
        VECSET ( BUFFER , VAL1 , VAL2 , LENGTH )                *)
entry VECSET
        mov     (msp)+ , r0      ; Count in r0.
        mov     (msp)+ , r1      ; Test value in r1.
        mov     (msp)+ , r2      ; Value to set in r2.
        mov     (msp)+ , r3      ; Buffer pointer in r3.
        add     r0 , r3          ; Point r3 to end of buffer.
9$:     cmpb    -(r3) , r1       ; Compare point to VAL2.
        bne     8$               ; If equal:
        movb    r2 , (r3)        ;  set to VAL1.
8$:     dec     r0               ; Decrement count.
        bne     9$               ; Loop back.
        next

(*      Returns the first value of three that is non-zero, or zero if all zero.
        IVAL := 3MAX ( VAL1 , VAL2 , VAL3 )              *)
entry 3MAX integer
        mov     (msp)+ , r1      ; VAL3 in r1.
        mov     (msp)+ , r0      ; VAL2 in r0.
        tst     (msp)            ; Test VAL1
        bne     5$               ; If non-zero, return VAL1.
        tst     r0               ; Else test VAL2.
        bne     4$               ; If zero:
        mov     r1 , (msp)       ;  return VAL3.
        br      5$               ; Else:
4$:     mov     r0 , (msp)       ;  return VAL2.
5$:     next

(*      Add a point to a blob.          *)
```

```
entry ADDBLOB
        mov     @ # ptr ( BLOB_REC ) , r0
        mov     @ # ptr ( FINAL_REC ) , r1
;  CURBLOB := BLBID
        mov     $o BLBID (r0) , $o CURBLOB (r1)
;  BOT ( CURPT ) := CURBLOB
        mov     $o CURPT (r1) , r2
        mov     @ # ptr ( LINES_REC ) , r3
        add     r2 , r3
        movb    $o CURBLOB (r1) , $o BOT (r3)
;  increment BLBTOT
        inc     $o BLBTOT (r0)
;  MAXY1 := CURLINE
        mov     $o CURLINE (r1) , $o MAXY1 (r0)
;  XBCOM := liadd ( XBCOM , CURPT )
        add     r2 , ( $o XBCOM + 2 ) (r0)
        adc     $o XBCOM (r0)
;  YBCOM := liadd ( YBCOM , CURLINE )
        add     $o CURLINE (r1) , ( $o YBCOM + 2 ) (r0)
        adc     $o YBCOM (r0)
        next

.end


(*      Blob records are assigned out of a pool of available space.
        FREEBLOBS is a stack, with pointer NXTBLOB, and, for redundancy,




        BLOBMAP is a bitmap of used blob records.          *)
define GETBLOB integer
  if ( NXTBLB ==0 ) print "OUT OF BLOBS" endif
  GETBLOB := FREEBLOBS ( NXTBLB )
  setbit ( GETBLOB BLOBMAP )
  mvzer ( BLOBS ( GETBLOB ) , sizew BLOB_REC )
  decrement NXTBLB
end


(*      Return a blob to the stack of free blobs.
        RETBLOB ( BLOB_ID )                                 *)
define RETBLOB
        integer ARG1
  increment NXTBLB
  FREEBLOBS ( NXTBLB ) := ARG1
  clrbit ( ARG1 BLOBMAP )
end


(*      Start a new blob.                                   *)
define NEWBLOB
 local
        integer TEMP1
  TEMP1 := GETBLOB              ; Get a new blob recorb
  with BLOBS ( TEMP1 )
    BLBID := TEMP1              ; Setup THIS ID
```

```
    MINX0 := CURPT
    MAXX1 := CURPT
    MINY0 := CURLINE
    ADDBLOB
 end

 (*      Merge two blobs into one blob.           *)
 define MERGEBLOB
  local
        integer TMPTOT
        long    TMPXCOM TMPYCOM
        integer TMPX0   TMPX1
        integer TMPY0   TMPY1
  if ( BLBID == CURBLOB )               ; Ring situation
        return
  endif
  TMPTOT := max ( CURBLOB BLBID )
  CURBLOB := min ( CURBLOB BLBID )      ; Merge into lower blob ID
  print #a ascii . , #z
  VECSET ( TOP , CURBLOB , TMPTOT , 256. )
  VECSET ( CEN , CURBLOB , TMPTOT , 256. )
  VECSET ( BOT , CURBLOB , TMPTOT , CURPT + 1 )
  with BLOBS ( TMPTOT )                  ; Save dying blob info in TEMP
        mvwds ( ptr ( BLBTOT ) , ptr ( TMPTOT ) , 9 )
        RETBLOB ( BLBID )
  with BLOBS ( CURBLOB )



        BLBTOT += TMPTOT
        XBCOM += TMPXCOM
        YBCOM += TMPYCOM
        MINX0 := min ( MINX0 , TMPX0 )
        MAXX1 := max ( MAXX1 , TMPX1 )
        MINY0 := min ( MINY0 , TMPY0 )
        MAXY1 := max ( MAXY1 , TMPY1 )
end

(*      Test the two points above the current point for a value.
        Returns TRUE if either has a value.          *)
define UPTEST integer
  UPTEST := max ( CEN ( CURPT ) , TOP ( CURPT ) )
  if ( UPTEST )
    with BLOBS ( UPTEST )
        ADDBLOB
  endif
end

(*      Test 2 points to the left in each of three rows for a value.     *)
define LEFTTEST
 local
        integer BLID
  do i 2
        BLID := 3MAX ( BOT ( CURPT - i ) CEN ( CURPT - i ) TOP ( CURPT - i ) )
```

```
        if ( BLID ) exit endif
    loop
    if ( BLID )
      with BLOBS ( BLID )
          MAXX1 := max ( MAXX1 , CURPT )
          ADDBLOB
    endif
  end


  (*      Test 2 points to the right in each of two rows for a value.     *)
  define RIGHTEST
   local
          integer BLID
    do 1 2
          BLID := max ( CEN ( CURPT + 1 ) , TOP ( CURPT + 1 ) )
          if ( BLID ) exit endif
    loop
    if ( BLID )
      with BLOBS ( BLID )
            if ( CURBLOB )0 )
                  MERGEBLOB
            else
                  MINX0 := min ( MINX0 , CURPT )
                  ADDBLOB
            endif
    endif




  end


  (*      Convert from point encoded format to blob format.       *)
  define PT2BLOB
   local
          integer TEMP1
    increment CURLINE               ; Get next raster line
    iter PTCNT                      ; Do for each point found in this raster line
      CURPT := POINTS ( 1 )         ; Set up current point.
      CURBLOB on                    ; No current blob at start.
      if ( not UPTEST )
          LEFTTEST
          RIGHTEST
      endif
      if ( CURBLOB (0 ) NEWBLOB endif   ; New blob found
    loop
  end


  (*      Set up the disagreement analysis.       *)
  define INITBLOB
    mvzer ( FINAL , sizew FINAL-REC )
    NXTBLB off
    CURLINE := BANDSIZE - 1
    do 1 , #BLOB - 1
```

```
        RETBLOB ( i' )                    ; Put all the blob-recs on the stack
   loop
   WNDOFF off
   MAPW ( WNDB )
   ptr ( LINES_REC ) := WNDADR
   mvzer ( TOP , BANDSIZE * 128. )
   ptr ( LINES_REC ) += BANDSIZE * 256. - 768.
 end


(*      Execute the analysis for a given number of lines.
        DOLINES ( #_LINES )                                    *)
define DOLINES
        integer #LINES
 local
        integer WBNDSZ  LASTPT
  WBNDSZ := 2/ ( BANDSIZE )
  LASTPT := 256. - BANDSIZE
  iter #LINES
        ptr ( LINES_REC ) += 256.
        mvzer ( BOT , WBNDSZ )
        mvzer ( BOT + LASTPT , WBNDSZ )
        RAS2PT ( BOT RASPTS LASTPT )
        PT2BLOB
        if ( NXTBLB < 10 ) BEEP print "TOO MANY BLOBS" ;; exit endif
  loop
end




(*      Perform the entire disagreement analysis.                    *)
define BLOB
 with M_EDGE
  ATTRG ( "MTCHIM" , 160000k )
  INITBLOB
  DOLINES ( 24. - BANDSIZE )
  do 64. , 832.
        WNDOFF := i
        MAPW ( WNDE )
        ptr ( LINES_REC ) := WNDADR + 1280.
        DOLINES ( 16. )
  loop ( 64. )
  WNDOFF := 896.
  MAPW ( WNDB )
  ptr ( LINES_REC ) := WNDADR + 1280.
  DOLINES ( 24. - BANDSIZE )
  mvzer ( BOT , BANDSIZE * 128. )
  DREGION
end

endfile

(*      CALCULATE THE CENTER OF MASS OF A BLOB.
        THE UN-NORMALIZED TOTALS ARE IN XBCOM AND YBCOM
        X,Y := BLBCOM ( BLOB-ID )                  *)
```

```
define BLBCOM long
        integer ARG1
  with BLOBS ( ARG1 )
   BLBCOM := wdlong ( lidiv ( XBCOM , BLBTOT ) , lidiv ( YBCOM , BLBTOT ) )
end
```

```
integer TOOMANY                          ; Flag to indicate too many defects.
integer COMTHRESH        DELTHRESH        ; Repeating defect threshholds.
integer TOTTHRESH        FLTTHRESH        ; Valid defect threshholds.
COMTHRESH := 3                   ; COM's may be within COMTHRESH pixels
DELTHRESH := 5                   ; DEL's may be within DELTHRESH pixels.
TOTTHRESH := 7                   ; Must have at least TOTTHRESH pixels.
FLTTHRESH := 2                   ; DELX and DELY at least FLTTHRESH + 1 pixels.

(*     Store the defects in the defect buffer.  (Primary mode)          *)
define STOREDEFS
  #_DFCTS off
  iter #BLOB
    if ( getbit ( i , BLOBMAP ) )
      with BLOBS ( i )
        if ( BLBTOT )= TOTTHRESH )
          if ( MAXX1 - MINX0 )= FLTTHRESH )
            if ( MAXY1 - MINY0 )= FLTTHRESH )
              with DEFECTS ( #_DFCTS )
                XCOM := lidiv ( XBCOM , BLBTOT ) - REG_X
                YCOM := lidiv ( YBCOM , BLBTOT ) - REG_Y
                DELX := 2/ ( MAXX1 - MINX0 + 1 )
                DELY := 2/ ( MAXY1 - MINY0 + 1 )
                increment #_DFCTS
                if ( #_DFCTS == MAX_DEFECT )
                        BEEP print "TOO MANY DEFECTS"
```

```
                    exit
                endif
              endif
             endif
            endif
          endif
        loop
      end


(*      Add a defect to the defect buffer, given a pointer to the DEFECT
        record to be added.
        ADD_DEFECT ( DEFECT_POINTER )                        *)
define ADD_DEFECT
        address DEFPTR
  with F_DEFCTS ( CUR_FRAME )
    mvwds ( DEFPTR , DEFECTS ( #_DFCTS ) , size DEFECT )
    increment #_DFCTS
    if ( #_DFCTS == MAX_DEFECT ) TOOMANY on endif
end


(*      Check every defect found against the previous defects to locate
        repeating defects.  (Confirm mode)                        *)
define CHECKREPT
 local
        DEFECT_BUFFER   PRIM_DEFS




        integer TXCOM   TYCOM
        integer TDELX   TDELY
 mvwds ( F_DEFCTS ( CUR_FRAME ) , PRIM_DEFS , sizew DEFECT_BUFFER )
 TOOMANY off
 #_DFCTS off
 with PRIM_DEFS
   iter #BLOB
     if ( getbit ( i , BLOBMAP ) )
       with BLOBS ( i )
         if ( BLBTOT )= TOTTHRESH )
           if ( MAXX1 - MINX0 )= FLTTHRESH )
             if ( MAXY1 - MINY0 )= FLTTHRESH )
                 TXCOM := lldiv ( XBCOM , BLBTOT ) - REC_X
                 TYCOM := lldiv ( YBCOM , BLBTOT ) - REC_Y
                 TDELX := 2/ ( MAXX1 - MINX0 + 1 )
                 TDELY := 2/ ( MAXY1 - MINY0 + 1 )
                 iter #_DFCTS
                  with PRIM_DEFS
                   with DEFECTS ( i )
                     if ( abs ( XCOM - TXCOM ) (= COMTHRESH )
                       if ( abs ( YCOM - TYCOM ) (= COMTHRESH )
                         if ( abs ( DELX - TDELX ) (= DELTHRESH )
                           if ( abs ( DELY - TDELY ) (= DELTHRESH )
                               ADD_DEFECT ( DEFECTS ( i ) )
                               exit
                           endif
                         endif
```

```
                    endif
                  endif
              loop
              if ( TOOMANY )
                    BEEP print "TOO MANY DEFECTS"
                    exit
              endif
          endif
        endif
      endif
    endif
  loop
end

(*      Store the defects found, whether primary of confirm mode.        *)
define STORE
  with M_EDGE
    ATTRC ( "IPSDBR" , 160000k )
    ptr ( IPSDE_REC ) := WNDADR
  with INSP_DATA_BASE
  with INSP_PLN
   with LAYERS ( MOD_LAYER )
    with DTL_LAYER_REV ( #_REVS - 1 )
     with L_RETICLE
      with RETICLE_DIE
       with D_PATTERNS ( MOD_PATTERN )




        with INSP_FR ( MOD_SITE )
         with F_DEFCTS ( MOD_FRAME )
          if ( I-MODE == PRIMARY )
                STOREDEFS
          else
                CHECKREPT
          endif
    DREGION
end
```

What is claimed is:

1. Apparatus for the automatic inspection of a semiconductor wafer surface comprising

means for illuminating the wafer surface,

scanning means for forming in a storage array a representation of the spatial distribution of illumination energy intensity reflected from the surface,

edge analysis means for automatically analyzing the reflected energy spatial distribution represented in said array for determining edge boundaries occurring on said wafer surface,

reference means for providing a reference pattern description of said wafer surface,

comparison means for comparing the edge boundaries determined by said analysis means with said reference pattern description for determining the location of boundary disagreements between the analysis means edge boundaries and the reference pattern description, and

means for generating an information output describing said boundary disagreements.

2. The apparatus of claim 1 wherein said illumination means comprises

dark field illumination means for illuminating said wafer surface with dark field illumination.

3. The apparatus of claim 2 wherein said scanning means comprises

a sensor array having a plurality of photoresponsive elements arranged in a linear pattern, each said element being responsive to illumination incident thereon,

means for mounting said linear sensor array for receiving energy reflected from said wafer surface,

means for focusing said reflected illuminated surface onto said sensor array elements,

means for reading from said sensor array and for storing data in said storage array corresponding to said spatial distribution representation.

4. The apparatus of claim 1 wherein said edge analysis means comprises

means responsive to said scanning means for generating a second spatial distribution representing local differences of the reflected illumination intensity across said wafer surface,

means responsive to said second spatial distribution for locating potential edge boundaries in said second spatial distribution,

means for storing said located potential edge boundaries when said potential boundaries have a strength which exceeds an edge threshold level, and

means for spatially filtering said located and stored edge boundaries for forming more continuous edge boundary patterns.

5. The apparatus of claim 4 wherein said storing means and said storage array are the same memory element.

6. The apparatus of claim 4 wherein

said illumination means comprises a dark field illumination means for illuminating said wafer surface with an oblique illumination from all directions, and

said generating means further comprises

means for spatially smoothing said first spatial distribution,

means for convolving said first spatial distribution along a first axis separately with a peak detecting spatial function and a step detecting spatial function,

means for convolving said first spatial distribution along a second axis orthogonal to said first axis separately with said peak detecting and said step detecting functions, and

means for generating from said orthogonal covolvutions said second spatial distribution.

7. The apparatus of claim 1 wherein said reference means comprises

a data reference source describing a wafer surface pattern, and

means for generating from said data source a data list of reference edge boundaries on said wafer surface.

8. The apparatus of claim 7 wherein said data source further comprises

an activity data source for identifying the spatial extent of active areas on said semiconductor wafer.

9. The apparatus of claim 7 wherein said reference means further comprises

means for identifying activity volumes on said semiconductor wherein a defect will adversely affect op ration of a circuit associated at least in part with said volume.

10. Th apparatus of claim 1 wherein said comparison means comprises

means for locating corresponding edg boundaries of said reference pattern description and said analysis means edge boundaries for effecting alignment of the reference and the analysis edge boundaries,

means for identifying non-corresponding edge boundaries of said reference pattern and said analysis means edge boundaries, and

disagreement means responsive to said identifying means for analyzing said identified non-corresponding edge boundaries for determining boundary disagreements on said wafer surface.

11. The apparatus of claim 10 wherein said disagreement means further comprises

means for classifying said boundary disagreements into a plurality of boundary disagreement classes.

12. The apparatus of claim 11 wherein one of said classes is a class of killer defects.

13. The apparatus of claim 10 wherein said identifying means further comprises

means for locating corner edge intersections in said reference pattern, and

means for providing a disagreement tolerance at said corner edge intersections for maintaining a correspondence between a squared reference corner and a rounded wafer corner.

14. The apparatus of claim 1 wherein said g nerating means comprises

means for selecting a boundary disagreement, and

means for repositioning said wafer surface for visual inspection of said wafer surface at said selected boundary disagreement.

15. The apparatus of claim 1 further wherein said wafer surface has a repeating reticle pattern thereon and said apparatus further comprises

means for automatically comparing boundary disagreements for at least two of said patterns to determine the presence of a repeating boundary disagreement, and

means responsive to a said repeating disagreement for classifying said repeating disagreement boundary as a reticle defect.

16. A method for the automatic inspection of a semiconductor wafer surface comprising the steps of

illuminating the wafer surface,

forming in a storage array a representation of the spatial distribution of illumination energy reflected from the surface,

automatically analyzing the reflected energy spatial distribution represented in the array for determining edge boundaries occurring on the wafer surface,

providing a reference pattern description of the wafer surface,

comparing the edge boundaries determined during said analyzing step with the reference pattern description and determining the location of boundary disagreements between the reference pattern description and the edge boundaries detected during the analyzing

step, and

generating an information output describing the boundary disagreements.

17. The method of claim 16 wherein said illuminating step comprises the step of illuminating said wafer surface with a dark field illumination for highlighting edge boundaries on said surface.

18. The method of claim 17 wherein said forming step comprises the steps of

mounting a linear sensor array element for receiving a said reflected energy from the wafer surface,

providing the array element with a plurality of photosensitive elements arranged in a linear pattern, each element being responsive to the illumination incident thereon,

focusing the reflected illuminated surface onto the array element, and

reading and storing signal values from said array element in said storage array, said signal values corresponding to said reflected energy spatial distribution.

19. The method of claim 16 wherein said analyzing step comprises the steps of

generating a second spatial distribution representing local differences of the reflected illumination across the illuminated wafer surface,

locating potential edge boundaries in said second spatial distribution depending upon said local differences,

storing the located potential edge

boundaries when a said boundary has a str ngth value which exceeds an edge threshold level, and

spatially filtering the located and stored edge boundaries for forming a more continuous edge boundary pattern.

20. The method of claim 19 further comprising the steps of

illuminating said wafer surface with a dark field illumination for highlighting edge boundaries on said surface, and

said generating step further comprises the steps of

spatially smoothing said first spatial distribution,

convolving said first spatial distribution along a first axis separately with a peak detecting spatial function and a step detecting spatial function,

convolving said first spatial distribution along a second axis orthogonal to said first axis separately with said peak detecting and step detecting functions, and

generating from said orthogonal convolutions said second spatial distribution.

21. The method of claim 16 further comprising the steps of

providing a data source for describing an expected wafer surface pattern, and

generating from the data source a list of reference edge boundaries properly expected to exist on said wafer surface.

22. The method of claim 21 wherein the providing step further comprises the step of identifying

the extent of active semiconductor areas on the semiconductor wafer.

23. The method of claim 21 wherein said providing step further comprises the step of identifying activity volumes of said semiconductor wherein a defect will adversely affect operation of a circuit associated at least in part with a said volume.

24. The method of claim 16 wherein the comparing step further comprises the steps of

locating corresponding edge boundaries of the reference pattern descirption and the analyzed edge boundaries on the wafer for providing effective alignment between the reference and analysis edge boundaries,

identifying non-corresponding edge boundaries of the reference pattern and the analysis edge boundaries, and

analyzing, in response to the identifying step, the identified non-corresponding edge boundaries for determining boundary disagreements on the wafer surface.

25. The method of claim 24 wherein the comparison step further comprises

classifying the boundary disagreements into a plurality of boundary disagreement classes.

26. The method of claim 25 wherein one of the classes is a class of killer defects.

27. The apparatus of claim 22 wherein the identifying step comprises the steps of

locating corner dge intersections in the reference pattern, and

providing a greater disagreement tolerance at the corner edge intersection before identifying a corner edge as a non-corresponding edge.

28. The method of claim 16 wherein said generating step comprises the steps of

selecting a boundary disagreemnt, and

repositioning the wafer surface for visual inspection of the wafer surface at the selected boundary disagreement.

29. The method of claim 16 wherein the wafer surface has a repeating reticle pattern thereon and the method further comprising the steps of

automatically comparing boundary disagreements for at least two repeating patterns to determine the presence of a repeating boundary disagreement, and

classifying any repeating boundary disagreement as a reticle defect.

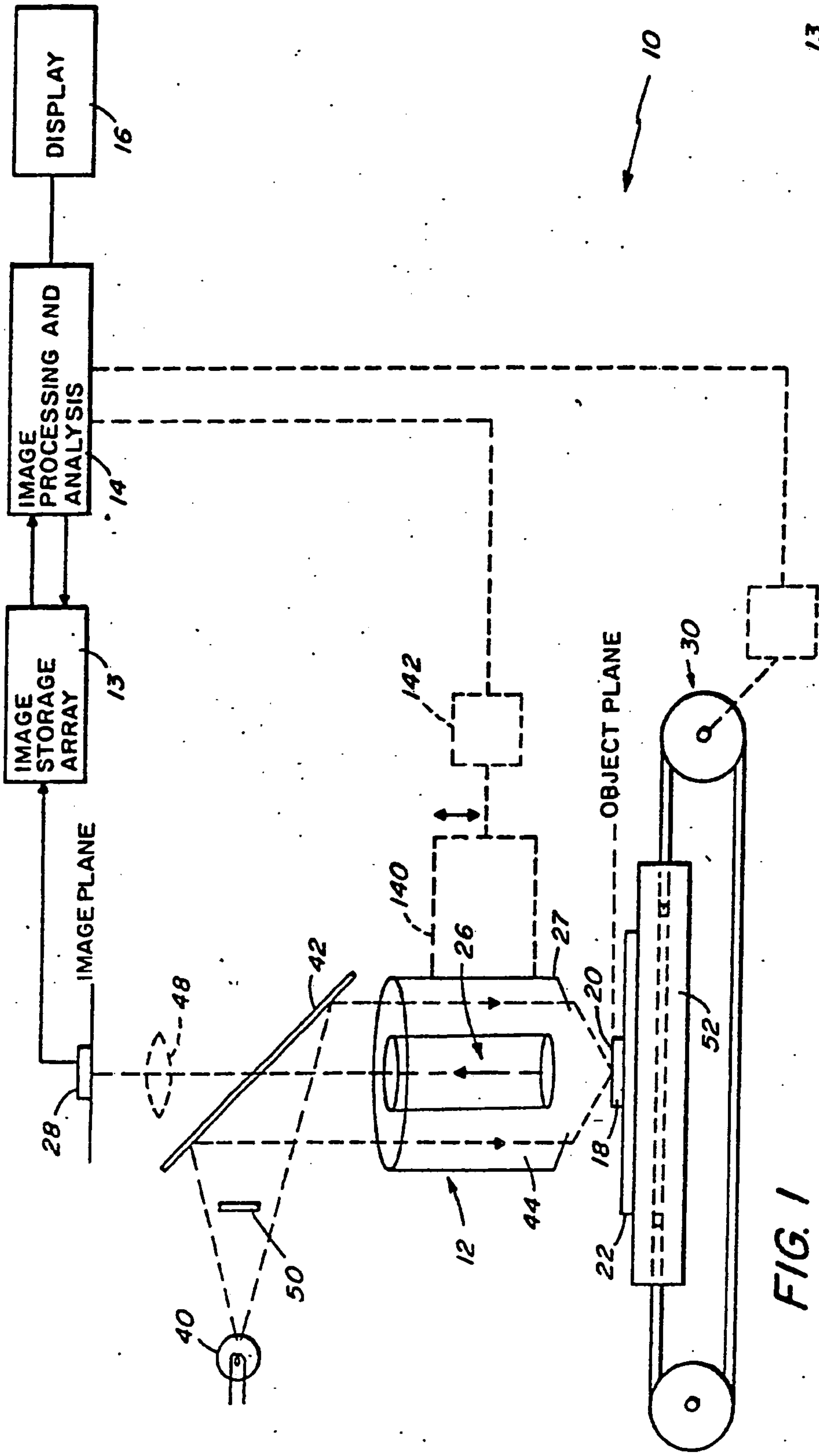

DISPLAY
16
IMAGE PROCESSING AND ANALYSIS
14
IMAGE STORAGE ARRAY
13
IMAGE PLANE
28
48
42
40
50
12
44
26
140
27
142
20
18
22
52
OBJECT PLANE
30
10

FIG. 1

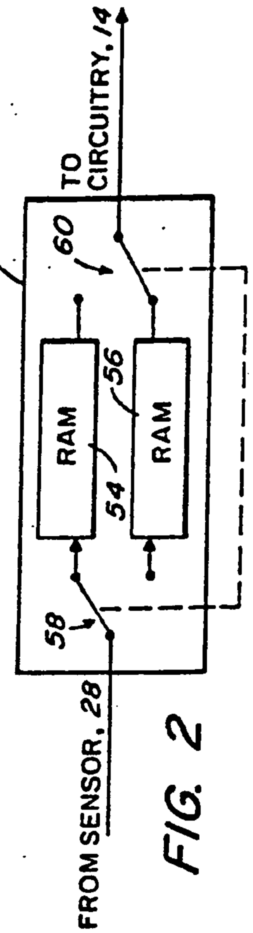

TO CIRCUITRY, 14
13
60
RAM
RAM
54
56
58
FROM SENSOR, 28

FIG. 2

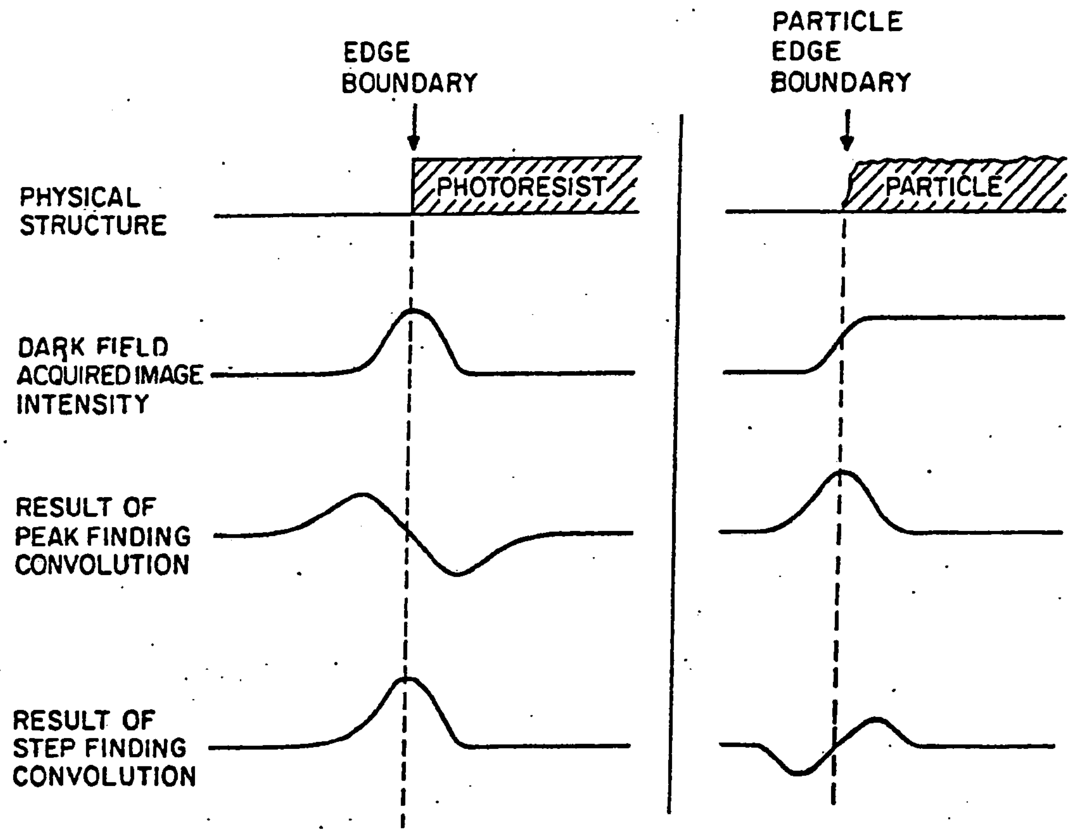

EDGE
BOUNDARY
PARTICLE
EDGE
BOUNDARY
PHYSICAL
STRUCTURE
PHOTORESIST
PARTICLE
DARK FIELD
ACQUIRED IMAGE
INTENSITY
RESULT OF
PEAK FINDING
CONVOLUTION
RESULT OF
STEP FINDING
CONVOLUTION

FIG. 3

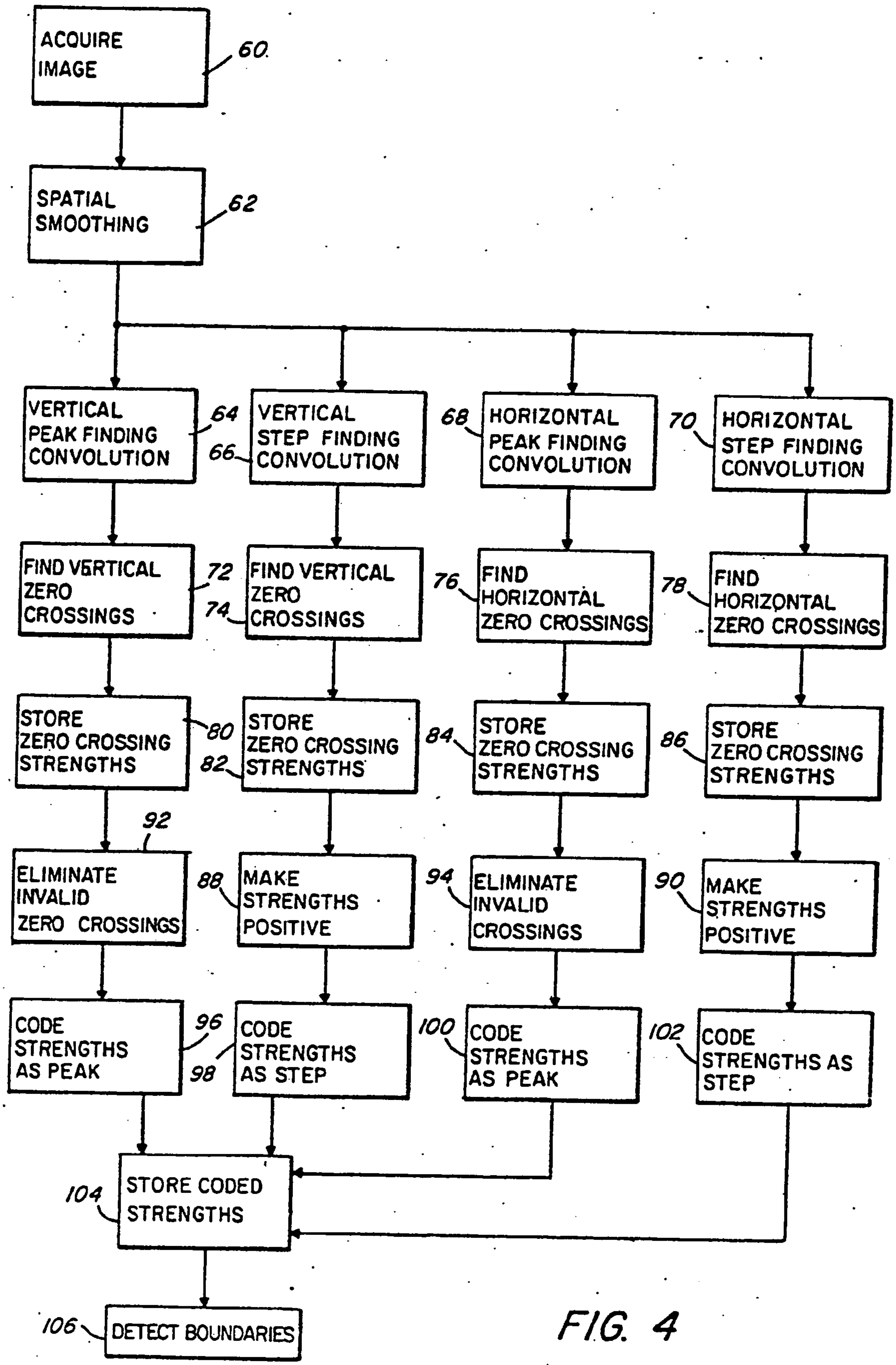

ACQUIRE IMAGE
60
SPATIAL SMOOTHING
62
VERTICAL PEAK FINDING CONVOLUTION
64
VERTICAL STEP FINDING CONVOLUTION
66
HORIZONTAL PEAK FINDING CONVOLUTION
68
HORIZONTAL STEP FINDING CONVOLUTION
70
FIND VERTICAL ZERO CROSSINGS
72
FIND VERTICAL ZERO CROSSINGS
74
FIND HORIZONTAL ZERO CROSSINGS
76
FIND HORIZONTAL ZERO CROSSINGS
78
STORE ZERO CROSSING STRENGTHS
80
STORE ZERO CROSSING STRENGTHS
82
STORE ZERO CROSSING STRENGTHS
84
STORE ZERO CROSSING STRENGTHS
86
ELIMINATE INVALID ZERO CROSSINGS
92
MAKE STRENGTHS POSITIVE
88
ELIMINATE INVALID CROSSINGS
94
MAKE STRENGTHS POSITIVE
90
CODE STRENGTHS AS PEAK
96
CODE STRENGTHS AS STEP
98
CODE STRENGTHS AS PEAK
100
CODE STRENGTHS AS STEP
102
STORE CODED STRENGTHS
104
DETECT BOUNDARIES
106

FIG. 4

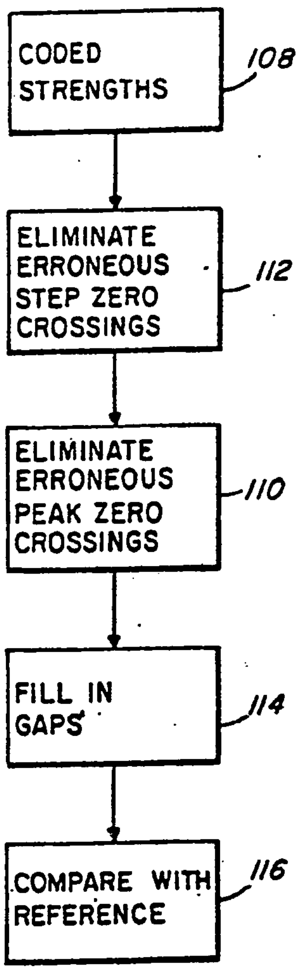


FIG. 5

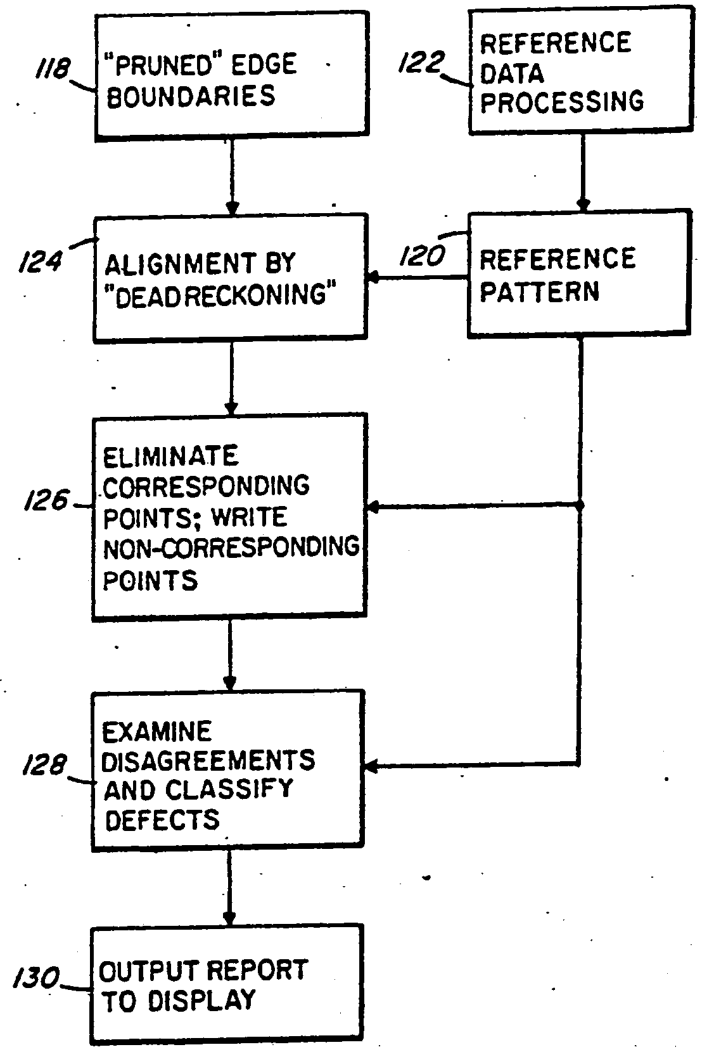


FIG. 6

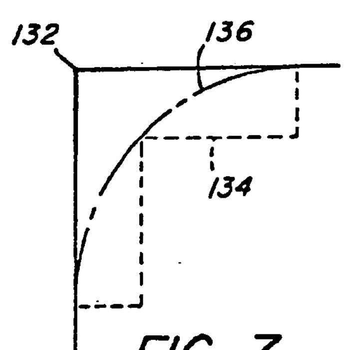


FIG. 7

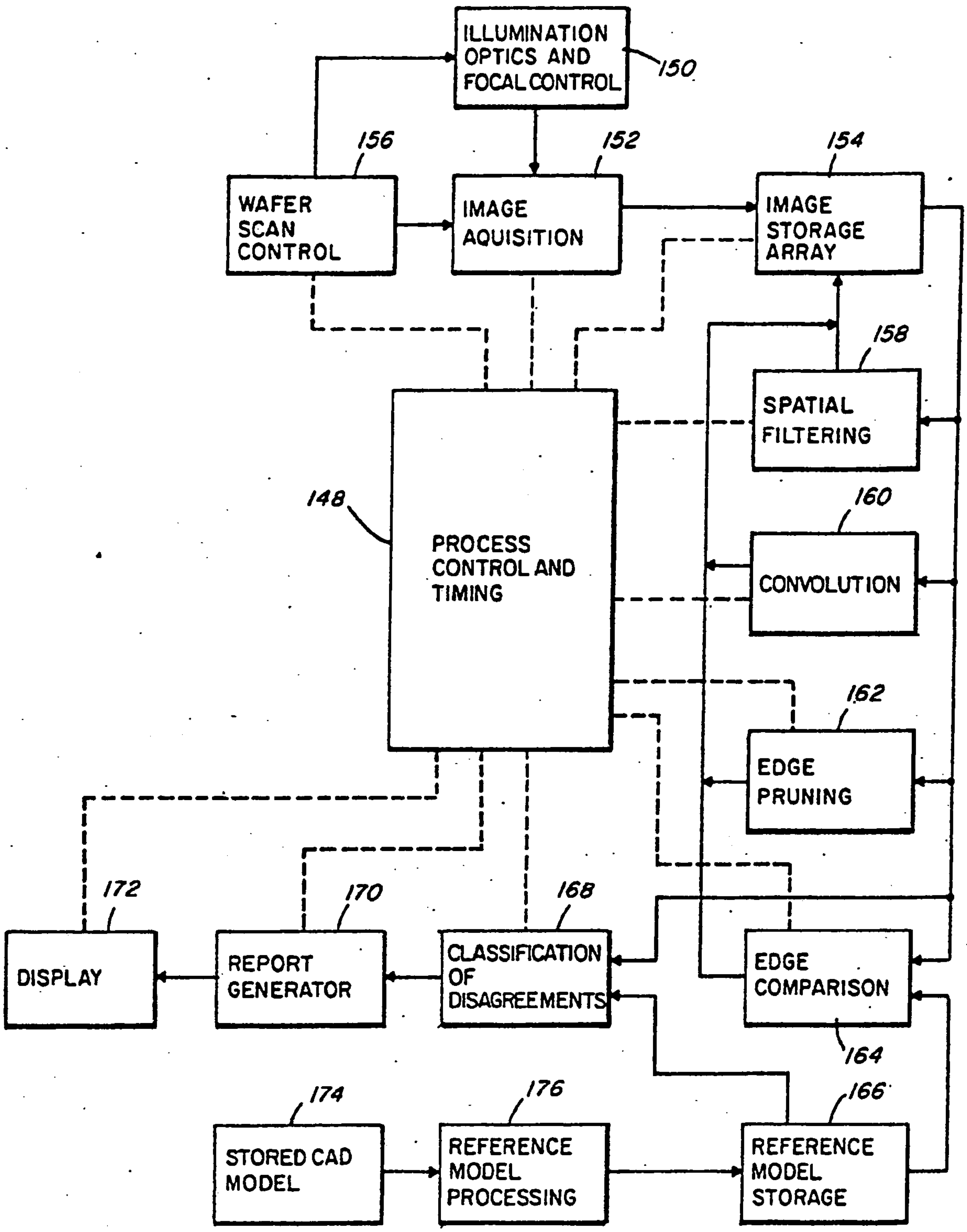

ILLUMINATION OPTICS AND FOCAL CONTROL
150
156
152
154
WAFER SCAN CONTROL
IMAGE AQUISITION
IMAGE STORAGE ARRAY
158
SPATIAL FILTERING
148
PROCESS CONTROL AND TIMING
160
CONVOLUTION
162
EDGE PRUNING
172
170
168
DISPLAY
REPORT GENERATOR
CLASSIFICATION OF DISAGREEMENTS
EDGE COMPARISON
164
174
176
166
STORED CAD MODEL
REFERENCE MODEL PROCESSING
REFERENCE MODEL STORAGE

FIG. 8

INTERNATIONAL SEARCH REPORT

International Application No PCT/US82/01277

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [3]

According to International Patent Classification (IPC) or to both National Classification and IPC

INT. CL.[3] G01B 11/00, G01N 21/47, Go6G 9/00
U.S. CL. 356/237, 394; 364/581, 728

II. FIELDS SEARCHED

Minimum Documentation Searched [4]

Classification System	Classification Symbols
U.S.	250/563; 356/237,394,400,446,448; 364/581, 728

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched [5]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [14]

Category *	Citation of Document, [16] with indication, where appropriate, of the relevant passages [17]	Relevant to Claim No. [18]
Y	US, A, 3,571,579, Published 23 March 1971, (Whitehouse et al)	6, 20
Y	US, A, 3,908,118, Published 23 September 1975, (Micka)	1-29
Y	US, A, 3,963,354, Published 15 June 1976, (Feldman et al)	1-29
Y	US, A, 4,240,750, Published 23 December 1980, (Kurtz et al)	1-29
Y	JP, A, 56-16,804, Published 18 February 1981, (Tsukazaki)	13, 27
Y	GB, A, 2,012,083, Published 18 July 1979, (Martinson)	6, 20
Y	N, IBM Technical Disclosure Bulletin, Vol. 13, #11, pp 3496, Published April 1971, Sommer et al, "Detection and Measurement of Epitaxial Spikes"	6, 20
Y	N, IBM Technical Disclosure Bulletin, Vol. 21, #6, pp 2336-7, Published November 1978, Grosewald et al, "Automatic Detection of Defects on Wafers"	1-29
Y	N, IBM Technical Disclosure Bulletin, Vol. 19, #2, pp 474-477, Published July 1976, Habegger, "Optical Determination of Semiconductor Device Edge Porfiles"	1-29

* Special categories of cited documents: [15]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search [2]	Date of Mailing of this International Search Report [2]
11 JANUARY 1983	21 JAN 1983
International Searching Authority [1]	**Signature of Authorized Officer [10]**
ISA/US	Wm. H. Punter William H. Punter

Form PCT/ISA/210 (second sheet) (October 1981)

International Application No. PCT/US82/01277

FURTHER INFORMATION CONTINUED FROM THE SEC ND SHEET

Y	N, Electronics, pp 44, 47, Published 04 December 1980, Waller, "Convolver on a Chip Pipelines Its Work"	6, 20

V.☐ OBSERVATIONS WHERE CERTAIN CLAIMS WERE FOUND UNSEARCHABLE [10]

This international search report has not been established in respect of certain claims under Article 17(2) (a) for the following reasons:

1.☐ Claim numbers ______, because they relate to subject matter [12] not required to be searched by this Authority, namely:

2.☐ Claim numbers ______, because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out [13], specifically:

VI.☐ OBSERVATIONS WHERE UNITY OF INVENTION IS LACKING [11]

This International Searching Authority found multiple inventions in this international application as follows:

1.☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims of the international application.

2.☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims of the international application for which fees were paid, specifically claims:

3.☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claim numbers:

4.☐ As all searchable claims could be searched without effort justifying an additional fee, the International Searching Authority did not invite payment of any additional fee.

Remark on Protest

☐ The additional search fees were accompanied by applicant's protest.

☐ No protest accompanied the payment of additional search fees.